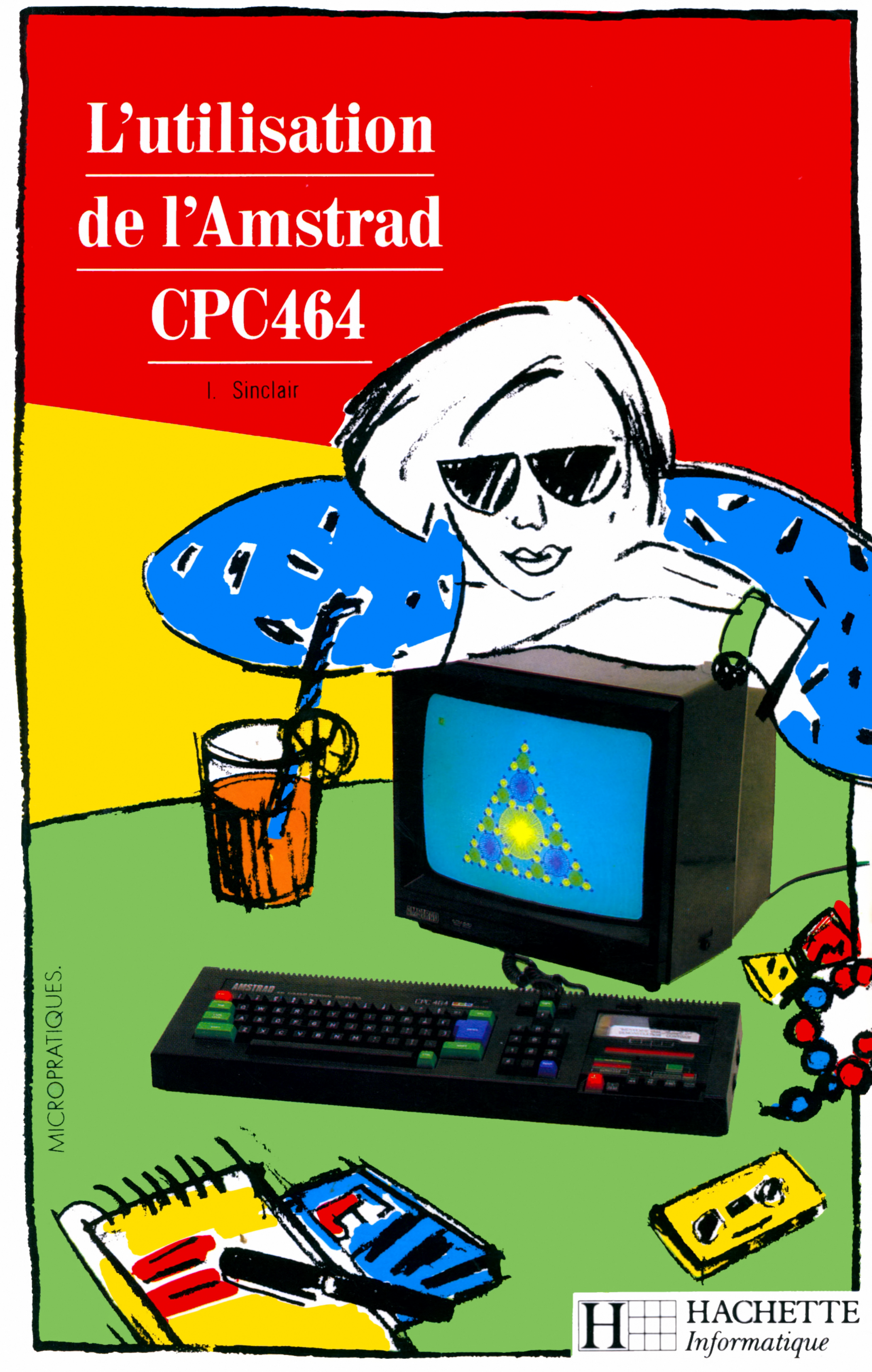

L'utilisation
de l'Amstrad
CPC464
I. Sinclair
MICROPRATIQUES.
AMSTRAD
HACHETTE
Informatique

Ian Sinclair

L'utilisation de l'Amstrad CPC 464

Traduit par
Alain Pierrot

HACHETTE
Informatique

Édition originale parue en langue anglaise sous le titre
Amstrad Computing
Publiée par Granada Publishing Ltd 1984
Réimprimée par Collins Professional and Technical Books 1984

Préface

L'ordinateur Amstrad CPC464 offre à l'utilisateur, qu'il soit particulier ou professionnel, bien plus qu'aucune machine précédente d'un prix comparable. Il offre même plus que beaucoup de machines d'un prix nettement supérieur. Par conséquent, beaucoup d'acquéreurs de ce micro n'auront jamais utilisé d'ordinateur auparavant, en tout cas pas un ordinateur doté de l'étendue de possibilités qu'offre le CPC464. Le manuel qui accompagne la machine est l'un des meilleurs qui aient jamais été édités pour un ordinateur domestique, mais c'est une tâche irréalisable que de satisfaire tous les besoins. En particulier, c'est une chose que d'apprendre comment utiliser un ordinateur, une autre que d'apprendre à le programmer pour son usage personnel, encore une autre que d'apprendre à concevoir ses propres programmes. Ce livre a pour but d'offrir précisément ce type d'aide progressive à l'acquéreur novice.

Il y aura fatalement des applications pour votre Amstrad CPC464 qui ne pourront être satisfaites par des programmes du commerce. Quand votre usage de l'informatique en sera venu au stade où vous devrez concevoir vos propres programmes pour répondre à vos propres besoins, vous devrez apprendre à programmer l'Amstrad CPC464 par vous-même. Le fait d'être programmé, après tout, est l'un des rôles dévolus à un ordinateur. Posséder un ordinateur sans le programmer par soi-même revient à acheter une voiture pour en confier la conduite à autrui. Si vous n'avez jamais programmé vous-même d'ordinateur, ce livre vous montrera comment le faire. Si vous avez quelque expérience d'anciens modèles d'ordinateurs, ce livre vous ouvrira tout un nouveau

monde de programmation. L'Amstrad CPC464 n'utilise pas la version simple du langage de programmation BASIC que l'on trouve sur les machines de conception antérieure — il faudra presque reprendre l'apprentissage à zéro si vous avez utilisé l'une de ces machines au préalable.

Je voudrais souligner le fait que ce livre a été écrit pendant que j'*utilisais* un Amstrad CPC464 qui avait été livré à domicile, et que les listings de programmes qui s'y trouvent proviennent directement d'une imprimante connectée à l'Amstrad CPC464. Cette remarque pourrait sembler vaine, mais on édite encore beaucoup de livres où les listings de programmes sont redactylographiés, et contiennent pour beaucoup des erreurs. Tous les programmes apparaissant dans ce livre, tous les exemples de commandes de programmation ont été testés sur l'Amstrad CPC464 qui se trouve devant moi. Rien n'a été repris du manuel sans vérification, et là où une commande n'a pas fonctionné clairement selon la description du manuel, j'ai souligné la différence. Tous les affichages sur écran que j'ai décrits ont été obtenus sur le moniteur couleur Amstrad. La raison en est que, à mon sens, beaucoup de lecteurs de ce livre utiliseront sans doute un tel moniteur à cause du prix remarquablement bas de l'ensemble ordinateur-moniteur. Toutefois, les lecteurs qui ont acheté l'ordinateur avec son alimentation séparée et l'adaptateur TV n'ont pas été oubliés.

1

La mise en œuvre du CPC464

Quand j'ai ouvert l'emballage de mon CPC464, j'ai constaté que plusieurs cabochons de touches étaient partis. Si cela vous arrive, ne vous inquiétez pas, car la plupart des cabochons se fixent en place par simple pression. L'exception est la longue barre d'espacement. Les trous des étriers attachés à cette barre doivent être accrochés aux deux tiges métalliques de la fente où elle se loge. Vous aurez à utiliser une petite pince pour y arriver. Une fois l'opération effectuée et le ressort mis en place, la barre d'espacement s'enfonce en place et votre CPC464 est prêt à l'action.

L'emballage du CPC464 contient le CPC464 lui-même, une cassette de démonstration, et un épais manuel. En plus de cet ensemble, vous aurez acheté l'un des trois éléments suivants : le moniteur à écran vert, le moniteur couleur, ou l'alimentation séparée avec l'adaptateur TV. Si vous avez acheté l'un ou l'autre des moniteurs, vous êtes prêt à travailler avec le CPC464. Si vous avez acheté l'alimentation/adaptation TV, toutefois, vous devrez aussi disposer d'un récepteur TV.
Le manuel montre très clairement comment les moniteurs ou l'alimentation doivent être connectés. Si vous utilisez l'alimentation, vous aurez aussi à connecter le CPC464 à la prise d'antenne de votre TV. Des conseils pour régler votre TV sur les signaux du CPC464 suivent dans ce chapitre.

Si vous utilisez les moniteurs, faites attention à la prise à six broches. Il ne faut pas la forcer dans la prise femelle, mais, sauf si le moniteur est au même niveau que l'ordinateur et très proche, vous risquez d'avoir un mauvais contact. J'utilisais le moniteur sur une étagère située au-dessus de l'ordinateur et j'ai dû allonger les câbles pour maintenir le contact de

la prise à six broches. Si vous constatez à l'allumage de la machine que le moniteur n'affiche rien, essayez de manipuler la prise. Vous ne ferez aucun dégât même si la prise ne fait plus contact en cours de programmation. Toutefois, si la prise d'alimentation secteur se détache, vous perdrez votre programme, à moins d'en avoir un enregistrement sur cassette. Aussi est-il très important d'utiliser l'enregistreur de cassettes incorporé. Vous trouverez aussi des conseils à ce sujet dans ce chapitre.

Tout d'abord, cependant, quelle que soit la version de la machine que vous ayez achetée, vous aurez à la munir d'une prise électrique car le CPC464 est livré avec un cordon du secteur nu.

Si vous avez l'habitude de monter vous-même des prises et ne faites pas appel aux services d'un électricien, le mieux est de choisir une prise munie d'un fusible de trois ampères. Le fil de terre restera inutilisé.

Une fois franchi cet obstacle, vous en êtes presque à mettre en route le CPC464, mais si vous utilisez l'alimention séparée, vous aurez besoin d'un téléviseur.

Un ordinateur est une machine qui envoie des signaux électriques qui peuvent servir à former des images sur un écran TV. Cela peut se faire de deux manières. L'une consiste à utiliser une forme spéciale d'affichage TV, conçue pour mettre directement à profit les signaux de l'ordinateur. C'est ce qu'on appelle un *moniteur* et d'ordinaire on ne peut recevoir avec lui des images TV reçues à l'aide d'une antenne. L'autre méthode consiste à convertir les signaux de l'ordinateur sous la même forme que les signaux envoyés par les émetteurs TV, de manière à pouvoir connecter l'ordinateur à la prise d'antenne d'un téléviseur ordinaire.

Il y a une importante différence entre téléviseur et moniteur. Bien qu'il soit commode — et économique — de pouvoir utiliser un téléviseur, l'image est de médiocre qualité. La raison en est qu'un téléviseur n'a jamais été conçu pour recevoir des signaux d'ordinateur, et qu'il faut traduire ceux-ci sous la forme de signaux comme ceux transmis par voie hertzienne. Il en résulte que les lettres et autres motifs paraissent flous sur l'écran, et que les couleurs paraissent décalées. Si vous utilisez le moniteur Amstrad, qui est d'un prix très raisonna-

ble par rapport aux autres, les motifs sur l'écran sont plus nets, et les couleurs bien meilleures. Le seul problème que j'aie rencontré avec le moniteur couleur Amstrad était que le contrôle de luminosité ne permettait pas d'avoir assez de lumière durant la journée. Si vous avez ce problème, ou un éventuel problème de luminosité excessive, renvoyez le moniteur pour un réglage interne. N'essayez en aucun cas d'enlever le boîtier du moniteur pour le régler vous-même, à moins d'être un expert en maintenance TV.

Obtenir l'image sur l'écran

A moins de connecter un téléviseur ou un moniteur au CPC464, vous ne pourrez voir ce que fait l'ordinateur. Il continuera tout aussi bien à calculer pour vous, mais vous ne verrez pas ce qui se passe. Pour connecter le CPC464 à un téléviseur, vous devrez relier la prise d'antenne de votre téléviseur et le boîtier d'alimentation. A moins que vous ne puissiez consacrer entièrement un téléviseur au CPC464, vous constaterez que vous devrez passer votre temps à brancher et débrancher le câble d'antenne et celui du CPC464. Ce n'est jamais une bonne chose à faire, parce que cela donne du jeu aux contacts de la prise. Vous trouverez une solution utile à ce problème dans le genre d'adaptateurs à deux voies illustré en *figure 1.1.* Ce genre d'accessoire vous permet de brancher une dérivation dans la prise d'antenne de façon à utiliser le téléviseur pour le CPC464 et pour les programmes télévisés sans avoir à manipuler les branchements.

Vous devez connecter le CPC464 à la TV ou à l'adaptateur en utilisant le câble spécial fourni avec l'alimentation du CPC464. Si le câble est trop court pour vous, vous pouvez acheter des rallonges et des raccords. Si vous avez utilisé l'adaptateur à deux voies, vous n'avez plus qu'à changer de canal pour passer de l'ordinateur à la télévision ! Si vous utilisez un moniteur, évidemment, vous n'avez qu'à vous assurer de la bonne insertion des prises.

Il n'est pas nécessaire que le téléviseur ou le moniteur que vous utilisez pour afficher les signaux du CPC464 soient en couleur, au moins au départ. Les activités de programmation du CPC464 ne nécessitent pas que vous voyiez les résultats en couleur, tant que vous n'en arrivez pas aux instructions qui concernent la couleur (cf. chapitre 8). Quand vous utilisez un

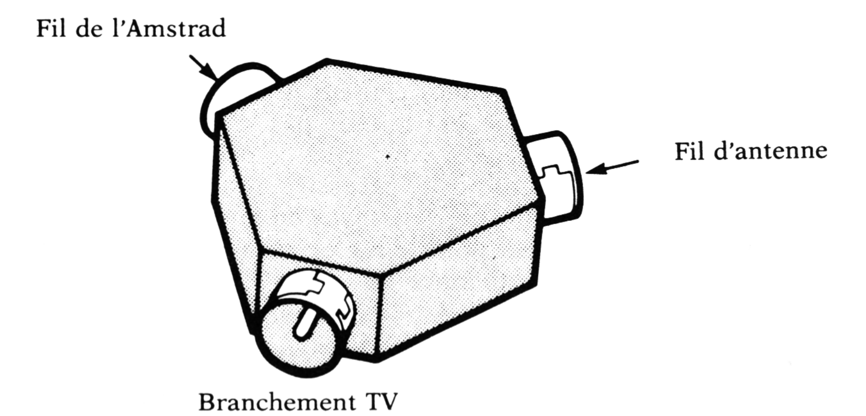


Fig. 1.1. Un adaptateur de câble d'antenne TV, qui vous permet de garder connectés à la fois l'ordinateur et l'antenne.

téléviseur noir et blanc pour afficher les signaux du CPC464, ou le moniteur monochrome, les couleurs sont marquées comme des nuances de gris, ou de vert, et restent tout à fait distinctes. Si vous utilisez un récepteur TV couleur, vous verrez les couleurs dans toute leur gloire, quoique tous les modèles de téléviseurs n'affichent pas des résultats d'égale qualité.

Le grand branchement

Maintenant, avant de brancher tout ce qui est en vue, et d'allumer, c'est une bonne idée de voir combien vous avez de prises électriques dans les environs, et où vous allez loger tout ce matériel. Quand vous utiliserez le CPC464 avec son moniteur, il vous suffira d'une prise de courant. Si vous utilisez l'alimentation séparée et un récepteur TV, vous aurez besoin de deux prises, une pour le CPC464 et une pour le téléviseur. La plupart des logements sont d'une pauvreté désespérante en prises de courant, aussi trouverez-vous utile d'acheter ou de fabriquer une rallonge portant une batterie de trois ou quatre prises femelles *(figure 1.2)*. Même si pour le moment vous pouvez vous contenter d'une seule prise, il y a des chances pour qu'elle ne se trouve pas là où vous en avez besoin. De plus, le CPC464 est un ordinateur si attrayant que vous voudrez probablement lui ajouter ultérieurement une imprimante et un lecteur de disquettes. Tous ces ajouts nécessiteront des prises de courant.

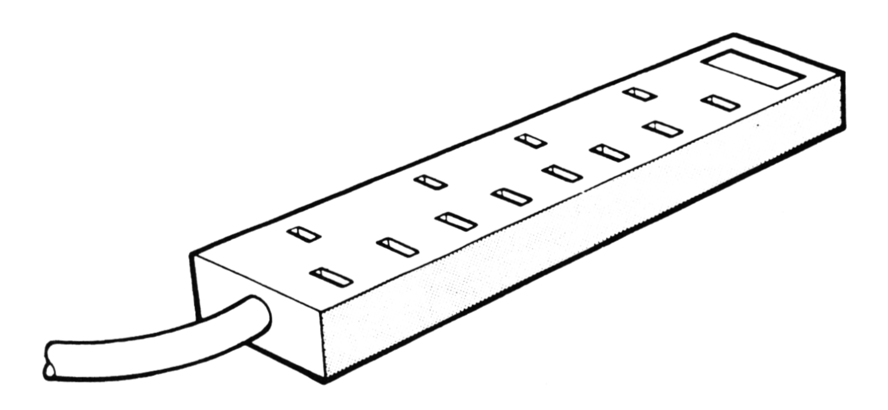


Fig. 1.2. Une batterie de quatre prises qui évite l'utilisation de vieux modèles de prises multiples.

Utiliser une batterie de prises évite beaucoup d'ennuis. Vous ne souhaitez pas envoyer votre CPC464 s'écraser par terre en marchant sur un câble. Ne vous fiez pas aux vieux modèles de prises multiples triplettes, ils ne donnent jamais un contact vraiment fiable. Le CPC464 a un interrupteur marche/arrêt, mais vous ne devriez jamais omettre de débrancher la prise de courant après une séance d'informatique. L'interrupteur du moniteur coupera aussi le courant de l'ordinateur, puisque le moniteur contient l'alimentation de l'ordinateur.

Quand vous avez l'équipement essentiel pour commencer à faire de l'informatique, c'est-à-dire le clavier du CPC464, l'alimentation et le téléviseur ou le moniteur, vous avez déjà besoin de beaucoup d'espace. Ultérieurement, vous voudrez probablement ajouter une imprimante, peut-être des lecteurs de disquettes, et autres suppléments qui font la différence entre un *système informatique* et la simple possession d'un ordinateur. Tout cela requiert de l'espace, et la meilleure méthode que j'aie trouvée pour organiser le matériel est l'un des postes informatiques spécialement conçus que l'on trouve dans le commerce *(figure 1.3)*. Si vous n'en êtes pas encore à ce stade, un bureau ou une table de bonne taille devront faire l'affaire pour l'instant. L'informatique est comme la hi-fi : vous pouvez toujours acheter quelque chose de plus !

Une fois tout logé et connecté, il vous faut maintenant pren-

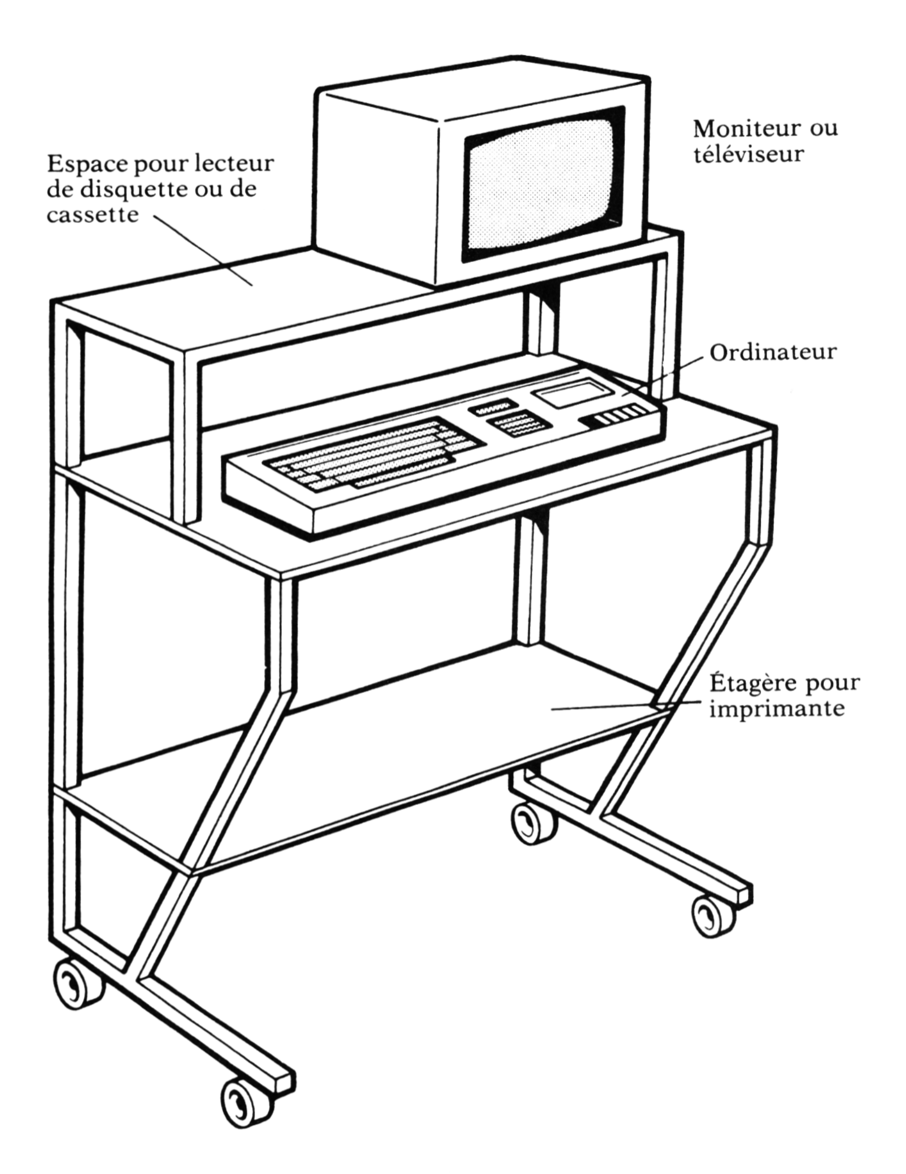


Fig. 1.3. Utilisation d'un poste Selmor pour loger tous les éléments d'un système CPC464.

dre quelques bonnes habitudes. Le CPC464, comme pratiquement tous les ordinateurs domestiques, utilise un lecteur-enregistreur de cassettes pour enregistrer l'information. Ceci parce que l'information stockée par l'ordinateur dans sa pro-

pre mémoire est perdue à chaque fois qu'on éteint la machine. L'enregistreur de cassettes peut enregistrer les signaux sur bande magnétique et les conserver après l'extinction de la machine. Quand ces signaux sont relus et restitués à l'ordinateur, la mémoire est à nouveau remplie par l'information, et l'ordinateur peut s'en servir. A la différence de la plupart des ordinateurs domestiques, le CPC464 a son lecteur enregistreur de cassettes incorporé. Cela présente plusieurs avantages : d'une part, vous n'avez pas à vous soucier de réglages délicats de niveau d'enregistrement, comme c'est souvent le cas quand on utilise un enregistreur de cassettes séparé, et il n'y a pas de connections supplémentaires à assurer ; d'autre part, cet enregistreur de cassettes est fait pour les signaux d'ordinateur, contrairement aux appareils ordinaires, et pour cette raison, il offre un moyen beaucoup plus sûr pour stocker de l'information sur bande magnétique. Toutefois, même la bande magnétique n'est pas idéale et il est toujours sage de faire deux copies de tout programme auquel vous tenez ; la seconde copie est appelée copie de secours ou copie *back-up*.

L'étape suivante est d'allumer le récepteur TV et le CPC464, ou le moniteur et le CPC464. Sauf si vous avez une chance exceptionnelle, ou si vous utilisez le moniteur, vous ne verrez sans doute rien apparaître sur l'écran. En effet, un *téléviseur* doit être réglé sur le signal du CPC464, tout comme on règle un poste à transistor pour recevoir les émissions de telle ou telle station, ou un téléviseur pour recevoir telle ou telle chaîne. Il faut dès lors régler la TV sur les signaux du CPC464.

Ce que vous cherchez, si le CPC464 n'a pas été touché depuis sa mise sous tension, est l'écran représenté dans le manuel page E 1.2. La seconde ligne de cet écran est un avis de copyright — sur mon CPC464 il portait la date de 1984. Quand vous voyez l'avis du copyright, tournez doucement le bouton dans un sens puis dans l'autre jusqu'à ce que vous obteniez le réglage où les mots sont vraiment nets ; Sur un téléviseur couleur les mots peuvent n'être jamais bien nets, rendez-les stables au moins, et aussi nets que possible.

Certains types de récepteurs TV noir et blanc et couleur plus anciens utilisaient des boutons poussoirs qui s'enclenchent avec un clic sonore quand vous les enfoncez. Il y en a d'ordinaire quatre, et vous devrez en utiliser un de disponible, ce

qui revient d'ordinaire à dire le quatrième. Poussez-le à fond ; le réglage se fait alors en le tournant. Essayez d'abord de l'amener en bout de course dans le sens contraire des aiguilles d'une montre. Ne vous étonnez pas du nombre de tours que vous pouvez faire avant d'arriver au bout. Si vous vous accordez sur le signal du CPC464 entre-temps, vous verrez le message sur l'écran. Si vous êtes arrivé en bout de course sans voir le signal, vous devrez repartir à la recherche dans l'autre sens, jusqu'au succès. En cas d'échec, vérifiez que vous avez correctement branché la prise d'antenne TV.

Les récepteurs modernes sont équipés de touches sensitives ou de petits boutons poussoirs pour sélectionner les canaux. Ces touches ne servent qu'à la sélection, pas au réglage sur un émetteur donné. Le réglage est assuré par un ensemble de petits boutons à tourner ou de petites molettes cachés derrière un panneau soit devant soit sur le côté du téléviseur. Les boutons ou les touches sensitives sont d'ordinaire numérotés et les numéros correspondants se retrouvent sur les boutons ou molettes à tourner. Utilisez le numéro le plus haut disponible (d'ordinaire 6 ou 12), appuyez sur la touche ou le bouton de ce numéro, et trouvez le bouton ou la molette correspondant. Le réglage se fait en tournant ce bouton ou cette molette. A nouveau, vous cherchez une image claire sur l'écran, et le silence sur le haut-parleur. Sur ce type de récepteurs, le réglage fin de l'image est d'ordinaire automatique une fois que le panneau cachant le système de réglage est fermé, par conséquent n'oubliez pas de le rabattre. Sinon, les circuits du récepteur qui assurent le maintien du réglage ne peuvent fonctionner, et vous verrez le réglage se modifier, ce qui vous obligera à passer votre temps à re-régler. Le CPC464 devrait fournir une bonne image sur pratiquement tous les téléviseurs. Si votre TV montre des défauts comme des sautes d'image, des couleurs très floues, vérifiez soigneusement la mise au point du réglage. Si les défauts persistent, et que la TV soit correctement réglée, vous devrez contacter le service après-vente de la TV — ou utiliser un autre modèle à l'avenir !

Si vous utilisez le *moniteur*, vous devriez avoir une bonne image sans pratiquement aucun réglage. La luminosité aura sans doute à être ajustée aux conditions d'éclairement de la pièce, mais il n'y a pas grand-chose d'autre à faire. Le moniteur couleur ne laisse accessible qu'un contrôle de verticalité (le contrôle d'horizontalité peut être atteint avec un tourne-

vis) mais il est très rare d'avoir à y toucher. Vous ne devriez tenter de régler ces points que dans le cas où l'image serait vraiment très instable.

Premiers contacts

Une fois que vous avez obtenu l'écran d'affichage de votre CPC464 commence le travail pour acquérir la maîtrise de son usage. Quand l'avis de copyright est affiché, vous verrez un carré brillant (jaune d'or) juste sous le mot 'READY'. Ce carré est appelé le *curseur*. Il est utilisé comme repère, et chaque fois que vous presserez une touche, une lettre (ou un nombre, ou ce qui est marqué sur la touche) apparaîtra à la position du curseur. Le curseur se déplacera ensuite à la position suivante sur l'écran. Il est important de noter à ce point que rien de ce que vous pouvez faire en appuyant sur les touches du clavier ne peut en aucun cas endommager le CPC464 — le pire qui puisse vous arriver est de perdre un programme qui se trouverait en mémoire. Par contre, vous pouvez endommager le CPC464 en renversant du café dessus, en le faisant tomber, ou en le connectant à d'autres circuits pendant qu'il est sous tension. Vous pouvez aussi brouiller des signaux sur vos cassettes si vous tentez de les sortir en cours d'enregistrement ou de lecture, ou si vous allumez ou éteignez l'ordinateur avec une bande en place. Éteignez *toujours* l'ordinateur, et tout ce qui lui est connecté quand vous touchez à l'une quelconque des prises qui sont à son dos. N'allumez ou n'éteignez *jamais* l'ordinateur avec une cassette en place, et n'enlevez *jamais* une cassette pendant qu'elle tourne. Gardez vos cassettes dans un endroit frais, bien à l'écart du moniteur ou de l'alimentation.

Le clavier

Il est temps désormais de jeter un coup d'œil au clavier, car le clavier est votre moyen de communiquer des instructions au CPC464. Si nous laissons de côté les touches du côté droit, la plupart des touches du CPC464 ressemblent à des touches de machine à écrire. La disposition des lettres et des chiffres est la même que celle d'une machine à écrire anglaise et, si vous en avez déjà utilisé une, en particulier une électrique, vous devriez vous y retrouver sur le clavier du CPC464 très rapidement. Quand vous appuyez sur l'une quelconque des

touches qui portent une lettre, vous voyez apparaître sur l'écran ladite lettre. Ce que vous voyez est la lettre *minuscule*, et non la *majuscule* (capitale). Tout comme celui d'une machine à écrire, le clavier du CPC464 donne normalement des minuscules. Si vous voulez une capitale, vous devez appuyer sur l'une des deux touches SHIFT en même temps que vous appuyez sur une lettre.

Les commandes que vous donnez à l'ordinateur sous forme dactylographiée peuvent être en minuscules ou en majuscules indifféremment. Si vous voulez rester en *capitales*, appuyez sur la touche marquée CAPS LOCK. Pour revenir en minuscules, il vous suffit de rappuyer sur cette touche CAPS LOCK. Malheureusement, il n'y a pas de témoin lumineux pour vous montrer la position de CAPS LOCK — il vous faut essayer pour voir ! Quelle que soit la position de CAPS LOCK, les touches qui portent deux symboles, comme la plupart des touches de chiffres, exigeront l'usage de SHIFT pour donner le symbole supérieur. Quand vous appuyez sur l'une de ces touches seule, vous obtenez le symbole marqué sur la partie inférieure de la touche. Utiliser SHIFT avec une de ces touches vous donne le symbole de la partie supérieure. Par exemple, si vous appuyez sur les touches 8 et SHIFT en même temps, vous obtenez (et si vous appuyez sur la touche 8 seule, vous obtenez 8.

A côté des touches de machine à écrire ordinaire, il y a un certain nombre de *touches spéciales*, qu'on ne trouve sur aucune machine à écrire. En haut à gauche du clavier, par exemple, vous trouverez une touche marquée ESC (*escape*, échappement) et à droite de la longue barre d'espacement vous trouverez une touche marquée CTRL (contrôle). Ces touches ESC et CTRL ont des usages soulignés dans le manuel. La touche ESC est particulièrement utile parce qu'elle vous permet d'arrêter le déroulement de n'importe quoi. Appuyer sur toute autre touche relancera le déroulement des opérations, tandis qu'appuyer à nouveau sur ESC rendra l'arrêt définitif. Quand un programme est interrompu de cette manière, vous pouvez toujours le relancer, parce que le programme est pas effacé de la mémoire.

Une autre manière d'arrêter le déroulement d'un programme consiste à appuyer simultanément sur CTRL, SHIFT *et* ESC, mais cette méthode a toujours pour effet d'effacer tout pro-

gramme de la mémoire. Nous en ferons usage plus tard, car c'est quelquefois le meilleur moyen de remettre l'ordinateur dans son état correct. Toutefois, vous devez vous assurer avant de prendre cette mesure que vous avez enregistré le programme que vous avez utilisé. La touche CTRL peut être utilisée sur afficher des motifs tout à fait différents des lettres qui sont marquées sur les touches — mais remettons cela à plus tard.

Vous devriez être mis au courant dès à présent d'une action, l'*effacement d'une lettre*. Tapez un mot et puis appuyez sur la touche DEL, qui se trouve en haut à droite du clavier. Vous verrez qu'une lettre disparaît à la fin du mot que vous avez tapé. Si vous maintenez la touche DEL enfoncée, vous verrez les autres lettres s'effacer aussi, tandis que le curseur se déplace vers la gauche. Quand il n'a plus de lettres à effacer, vous entendrez un signal sonore du haut-parleur incorporé du CPC464. Augmentez le réglage de volume (à côté de l'interrupteur marche/arrêt sur le flanc droit de l'appareil) si vous n'entendez pas le signal. Cette action d'effacement de DEL est l'un des moyens les plus utiles que vous ayez pour effacer les fautes de frappe.

L'ensemble de quatre touches marquées de flèches est l'ensemble des *touches de curseur*. Essayez-les, et vous verrez qu'elles font déplacer le curseur dans la direction de la flèche marquée sur chacune. Ces touches sont utilisées conjointement avec la touche COPY au centre du groupe pour un type d'édition. Cette technique est expliquée en détail dans l'*appendice A*. Vous pourrez vous y reporter en temps opportun. Finalement, l'ensemble de douze touches en bas à droite du clavier comprend ce qu'on appelle les touches de *bloc numérique*. Pour ceux d'entre vous qui sont droitiers, ces touches sont censées rendre l'entrée de nombres plus faciles.

La touche spéciale la plus importante de toutes, cependant, en ce qui nous concerne pour l'instant, est la grande touche marquée ENTER. Elle est placée comme la touche de retour à la ligne (retour chariot) d'une machine à écrire électrique, mais son action n'est pas exactement similaire. Appuyer sur la touche ENTER signale à l'ordinateur que vous avez fini de taper une instruction et que vous voulez que l'ordinateur lui obéisse maintenant. Si vous avez l'habitude d'utiliser une machine électrique, vous aurez à modifier quelques-unes de

vos habitudes en ce qui concerne cette touche. Sur une machine à écrire, vous appuieriez sur la touche retour-chariot chaque fois que vous voulez revenir à la ligne, pour commencer à écrire à gauche de la nouvelle ligne. La touche ENTER de l'ordinateur fait bien plus que ça. Si ce que vous tapez sur le CPC464 prend plus qu'une ligne d'écran, la machine passera *automatiquement* à la ligne suivante pour vous. La touche ENTER *ne doit pas* être utilisée à cet effet. La touche ENTER ne sert que lorsque vous voulez que la machine exécute un ordre ou enregistre une instruction, et non lorsque vous voulez simplement passer à la ligne. Toutefois, quand vous appuyez sur ENTER, l'ordinateur vous donnera une nouvelle ligne, et placera le curseur au début de celle-ci. La position où une lettre (ou un autre caractère) apparaîtra quand vous appuierez sur une touche est indiquée par le curseur.

Vous constaterez que l'action de chaque touche se répète si vous maintenez votre doigt dessus, et que cette répétition est très rapide. Si vous tapez une suite de lettres sans signification et puis ENTER, l'ordinateur répond d'ordinaire par le message :

Syntax error

(erreur de syntaxe) qui est suivi, à la ligne suivante, par le mot 'Ready' (Prêt) et le curseur. Le message d'erreur de syntaxe intervient parce que l'ordinateur est une machine simple : il ne peut reconnaître que quelques mots — les mots qu'on appelle *mots réservés* ou *mots d'instruction*. Si ce que vous tapez ne contient pas ces mots ou utilise ces mots de manière incorrecte, il y a une erreur de syntaxe, du moins du point de vue de l'ordinateur. Cela peut signifier quelque chose pour vous, mais cela ne veut rien dire pour l'ordinateur ! Le mot 'syntaxe' signifie la manière dont les mots sont utilisés dans n'importe quel langage, et ce que votre ordinateur utilise comme instructions est une sorte de langage.

L'utilisation des cassettes

L'ordinateur CPC464, comme tous les autres, stocke des instructions et des informations dans sa mémoire — mais seulement quand la machine est allumée. Chaque fois que vous éteignez votre CPC464, tout ce qui était stocké dans sa mémoire est instantanément perdu, et vous ne pouvez le

récupérer, même en le rallumant très vite. Aussi devons-nous stocker les programmes et l'information dont nous avons besoin pour utiliser les programmes sous une autre forme. Les formes les plus utiles sont les disques magnétiques et la bande magnétique. Un système de disquettes magnétiques est désormais disponible pour le CPC464. Un système de disquettes offre l'avantage d'être beaucoup plus rapide et beaucoup plus fiable que des cassettes.

L'ordinateur a des circuits qui convertissent les instructions d'un programme en signaux qui peuvent alors être enregistrés sur la bande d'un enregistreur de cassettes. Quand ces signaux sont relus, un autre ensemble de circuits reconvertit les signaux sous forme de programme. De cette manière, l'usage du système de cassettes du CPC464 vous permet d'enregistrer vos programmes sur bande magnétique et de les relire. Avant d'attaquer le reste de ce livre, il est important de vérifier que vous savez utiliser l'enregistreur de cassettes pour enregistrer et relire des programmes.

Le plus facile est d'essayer d'abord d'utiliser la cassette pour mettre un programme en mémoire dans l'ordinateur. Cette opération est appelée *chargement*, et puisque votre CPC464 est livré avec un ensemble de programmes sur cassette, ils offrent un moyen commode d'acquérir de l'expérience. Vous verrez dans les notes qui accompagnent les programmes que vous ne pouvez pas facilement les copier, parce qu'ils ont été protégés contre la copie pour vous en empêcher. Normalement, vous ne voudriez pas acheter de programmes protégés ainsi contre la copie, parce que vous ne pouvez en faire de copie de sécurité. Il est essentiel de faire une copie de sécurité de tout programme important pour vous. Cela ne signifie pas que le système de cassettes est particulièrement peu fiable ; c'est simplement une précaution normale. Tous les systèmes d'enregistrement sont sujets à la perte de signaux, et les gens qui utilisent des ordinateurs gardent toujours plusieurs copies de toute information précieuse. Il est, à vrai dire, plutôt déraisonnable de payer très cher un programme que vous ne pouvez copier. Si un vendeur de programmes annonce fièrement que son programme ne peut être copié, *ne l'achetez pas !* De toute manière, si vous avez la moindre expérience des cassettes, vous pourrez faire une copie de n'importe quelle bande en utilisant un enregistreur à double platine, ou deux enregistreurs reliés par un câble adéquat.

Chargez la cassette de « Bienvenue » qui accompagne votre ordinateur, en suivant les très claires instructions du manuel. Quand vous entendez l'arrêt de la bande, appuyez sur la touche STOP de l'enregistreur. C'est important : bien que l'ordinateur arrête le moteur qui entraîne la bande, le cylindre qui communique le mouvement à la bande est encore tenu appliqué contre la bande par une poulie de caoutchouc, qu'on appelle galet presseur. Si ce galet presseur est laissé serré contre le cylindre (ou cabestan) pendant longtemps, il se déformera en formant un méplat sur un côté, et cela peut être à l'origine d'ennuis d'enregistrement et de lecture. Quand cela arrive, la réparation n'est pas si simple qu'elle le serait pour un enregistreur séparé. Vous devez porter l'ordinateur tout entier chez un réparateur parce que l'enregistreur est incorporé et ne peut facilement être isolé. Quand vous appuyez sur la touche STOP de l'enregistreur, le programme se mettra en route. Seules certaines bandes enregistrées par les fabricants et par les fournisseurs de logiciels professionnels devront être chargées ainsi. Le point important à apprendre maintenant est comment enregistrer vos propres programmes.

Enregistrer votre travail

Avant de faire un enregistrement pour essayer le système, il vous faut un programme à enregistrer, et cela implique un peu de dactylographie. Voilà qui est aisé si vous venez d'allumer le CPC464, mais si vous avez pianoté au hasard, c'est une bonne idée que d'éteindre puis d'allumer, après avoir enlevé la cassette.

Frappez le nombre 10 (1 et puis 0), et ensuite le mot rem. Peu importe que vous tapiez rem ou REM. C'est une instruction reconnue par l'ordinateur, et peu importe que vous la tapiez en majuscules ou en minuscules, l'ordinateur va (ultérieurement) la convertir en majuscules. Vérifiez que votre texte paraît correct, et appuyez sur ENTER. L'effet de cette action est de placer la ligne d'instruction '10 REM' dans la mémoire du CPC464.

Quand vous frappez le premier chiffre, le caractère apparaît à l'écran à la place du curseur. Quand vous appuyez sur ENTER, le curseur passe à la ligne suivante. En même temps, votre instruction reste où elle a été tapée sur l'écran. Si vous

avez utilisé des minuscules, vous verrez encore la ligne avec ses minuscules, telle

10 rem

Si vous voyez une faute pendant que vous tapez la ligne, utilisez l'action du retour arrière avec effacement, en appuyant sur la touche DEL. Cela ramène le curseur en arrière, et efface la lettre sur laquelle le curseur se trouve désormais. Vous pouvez alors taper la lettre correcte, qui remplacera l'erreur. Si cela rend la ligne correcte, appuyer sur ENTER introduira la ligne dans l'ordinateur. Si vous avez tapé quelque chose d'incorrect comme REN ou RXM vous ne serez absolument pas prévenu — l'ordinateur accepte tout ce que vous pouvez taper après un nombre.

Pour corriger une ligne telle que :

20 Ren

vous pouvez, au lieu d'utiliser DEL, vous contenter de taper la version correcte :

20 rem

et appuyer sur ENTER. Maintenant, tapez le reste des lignes, comme illustrées sur la *figure 1.4,* sans oublier d'appuyer sur ENTER chaque fois que vous avez fini de taper une ligne.

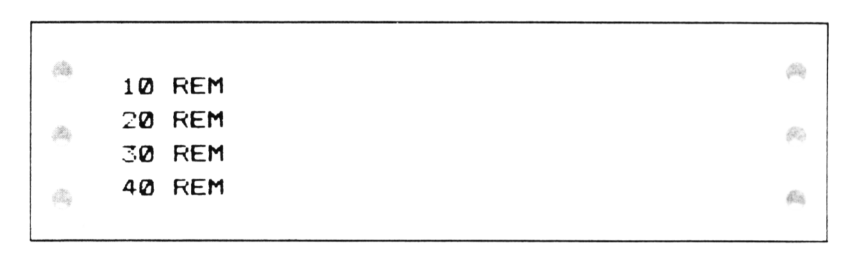


Fig. 1.4. Un programme pour tester les techniques d'enregistrement et de lecture sur cassette.

Les nombres sont appelés *numéros de lignes,* et ils sont présents pour deux raisons. La première est de signaler à l'ordinateur qu'il s'agit d'un programme ; la seconde est de le guider, car l'ordinateur effectuera normalement les instructions dans l'ordre des numéros de lignes. Vous pouvez vérifier que votre programme a l'air correct en demandant à l'ordinateur de le *lister.* 'Lister' signifie que l'ordinateur imprime sur l'écran tout ce que vous avez stocké dans sa mémoire. Utiliser

les numéros de lignes assure l'enregistrement en mémoire des instructions, et si vous tapez 'list' et appuyez sur ENTER, vous verrez votre programme. Ne soyez pas surpris de constater que toutes les lettres minuscules (comme rem) ont été converties en majuscules (comme REM), car c'est une partie du rôle de l'ordinateur, avec la mise en ordre des numéros de lignes, et l'insertion d'un espace entre le numéro et la première lettre de l'instruction. Vérifiez sur ce 'listing' que le programme est identique à la version imprimée sur la *figure 1.4.*

Maintenant assurez-vous que vous avez une cassette prête. Choisissez une cassette neuve, de préférence une C12 ou une C15 pour usage informatique. Des cassettes convenables se trouvent chez votre revendeur informatique. Placez la cassette dans l'enregistreur (le manuel montre très clairement l'opération page E 1.6), et appuyez sur la touche REW (rembobinage) pour vous assurer que la cassette est complètement rembobinée. Quand vous en êtes sûr, appuyez sur le petit bouton du compteur pour le remettre à zéro. Maintenant appuyez sur la touche FF (fast forward, bobinage rapide) de l'enregistreur, et arrêtez la bande quand le compteur est à 002. Cela garantit que vous avez un morceau de bande vierge pour enregistrer, *et c'est très important.* Au début de chaque cassette il y a toujours un morceau de bande plastique sans revêtement magnétique. On l'appelle l'*amorce,* et si vous tentez d'enregistrer sur cette partie de la cassette, votre tentative aboutira à un échec ! De plus, le début de la véritable bande à enregistrer est souvent légèrement froissé ou marqué. En dépassant cette portion, vous êtes bien plus assuré d'obtenir un enregistrement de bonne qualité.

Et maintenant pour réaliser l'enregistrement. D'abord vous devez taper :

SAVE "TEST"

et appuyer sur la touche ENTER. Le mot SAVE est l'instruction signifiant à l'ordinateur que vous voulez sauver (enregistrer) un programme sur une cassette. Le mot TEST est un *nom de fichier* que l'ordinateur utilisera pour identifier le programme si on le lui demande. Il est possible de sauver un programme sans nom de fichier, du moment que vous tapez SAVE et les premiers guillemets ("), mais vous ne devriez pas prendre cette habitude. Normalement vous conserverez plu-

sieurs programmes différents sur chaque cassette, et vous aurez besoin des noms de fichiers pour demander à l'ordinateur de récupérer celui que vous voulez. Quand vous appuierez sur la touche ENTER, vous verrez :

Press REC and PLAY then any key :

ce qui signifie « Appuyez sur REC (enregistrement) et PLAY (marche) puis sur n'importe quelle touche ». Vous devez donc appuyer sur les touches REC et PLAY de l'enregistreur jusqu'à ce qu'elles s'enclenchent. Maintenant le message "press any key" (appuyez sur n'importe quelle touche) est presque correct, mais pas tout à fait. En effet, les touches SHIFT, CAPS LOCK et CTRL n'auront aucun effet, et la touche ESC arrêtera tout le processus. Vous pouvez cependant appuyer sur la barre d'espacement ou ENTER ou n'importe quelle lettre (ou chiffre). Quand vous frappez une de ces touches, le moteur du lecteur de cassettes se met en marche et vous verrez bientôt un message apparaître sur l'écran. Dans le cas présent, le message sera :

Saving TEST block 1

(qui signifie « En cours de sauvegarde TEST bloc 1 »), et si le volume du haut-parleur est réglé assez haut vous pourrez entendre les légers sons que font les signaux représentant votre « programme ». Quand le programme a été enregistré, le moteur de l'enregistreur s'arrête, et le mot "Ready", suivi du curseur, réapparaît sur l'écran. Le programme est maintenant enregistré, et la taille du programme est indiquée par le nombre qui suit le mot "block". Pour la plupart, de courts programmes comme ceux utilisés dans ce livre n'auront besoin que d'un bloc pour être enregistrés. Le moteur de l'enregistreur de cassettes s'arrêtera automatiquement en fin de processus, mais il vous reste encore à appuyer sur la touche STOP de l'enregistreur pour relâcher la bande. C'est tout ce qu'il y a à faire pour enregistrer.

Et maintenant voici le bouquet. Vous devez vous assurer que l'enregistrement a marché. Tapez NEW et appuyez sur ENTER. Ceci devrait avoir effacé votre programme de la mémoire. Juste pour l'élégance de l'affichage, tapez CLS et appuyez sur ENTER. Cette commande nettoie l'écran, si bien qu'il n'y aura pas de confusion. Maintenant tapez LIST et appuyez sur ENTER. Rien ne devrait apparaître — LIST signifie « afficher à l'écran une liste des instructions de pro-

gramme », et il ne devrait pas y avoir de programme en mémoire ! Une autre solution pour faire tout cela consiste à appuyer simultanément sur CTRL, SHIFT et ESC pour nettoyer mémoire et écran.

Vous pouvez maintenant charger les instructions en provenance de la bande. Tapez LOAD « test » et appuyez sur ENTER. Vous obtiendrez là encore un message, cette fois-ci d'appuyer sur la touche PLAY (vous *ne devez pas* appuyer sur la touche REC) et puis sur n'importe quelle touche. Quand vous le ferez, le moteur de la cassette se remettra en marche et vous verrez vite apparaître un message à l'écran. Cette fois, le message est :

Loading TEST block 1

(En cours de chargement TEST bloc 1) et si le volume est assez haut, vous entendrez à nouveau les sons. Encore une fois, quand le programme est chargé, le moteur de l'enregistreur s'arrêtera, et vous devrez appuyer sur la touche STOP de l'enregistreur. Tapez LIST maintenant, et puis appuyez sur ENTER. Vous devriez voir apparaître votre programme sur l'écran.

Une fois que vous savez convenablement sauver des programmes sur bande et les recharger, vous pouvez vous mettre à programmer en confiance. Une fois que vous avez passé une heure ou plus à taper un programme sur le clavier, il est bon de savoir que quelques minutes de travail de plus préserveront sur bande votre effort, de manière à ce que vous n'ayez pas à tout retaper. Le système de cassettes du CPC464 est un

---

1. Gardez les cassettes au sec, au frais, sans condensation.
2. Stockez-les et utilisez-les bien à l'écart des aimants. Cela comprend les haut-parleurs, les téléviseurs et les moteurs électriques.
3. Ne touchez jamais à la bande avec vos doigts.
4. Si la bande se coince, sortez la cassette, frappez-la sur votre main, et essayez de la rembobiner rapidement à fond dans chaque sens.
5. N'utilisez jamais le début de la bande d'une cassette.
6. Ne gardez jamais une bande trop longtemps sans la dérouler ou au moins la rembobiner à vitesse rapide.

Fig. 1.5 Un mémento pour assurer une longue vie à vos cassettes.

système très fiable, bien que les programmes prennent plutôt plus de temps à sauver ou charger que sur d'autres systèmes. La seule précaution à prendre est le soin des cassettes. Le tableau de la *figure 1.5* devrait vous servir d'aide-mémoire à ce sujet.

Plus tard, lorsque vous aurez à faire à des programmes bien plus longs, vous trouverez probablement le temps passé à utiliser une cassette plutôt irritant. La réponse définitive à cette irritation réside dans le passage au système à disquettes, mais vous *pouvez* obtenir des opérations plus rapides sur cassettes ! Si vous tapez :

SPEED WRITE I

et puis appuyez sur ENTER, cela bascule le système de cassettes sur une vitesse d'enregistrement double. La bande défile au même rythme, mais l'enregistrement se fait plus rapidement. Vous n'avez pas besoin d'utiliser cette commande au chargement — l'ordinateur sélectionnera automatiquement la vitesse correcte. Ce n'est pas aussi sûr que la vitesse lente, mais si vous prenez bien soin de vos cassettes, la différence de fiabilité n'est pas sensible. La différence de rapidité, en revanche, est très sensible quand vous utilisez de longs programmes. C'est une bonne idée que de garder une copie de sécurité d'un long programme enregistrée à vitesse lente, et d'utiliser une copie d'usage « quotidien » qui a été enregistrée à vitesse rapide. Pour revenir à la vitesse normale (lente), tapez juste :

SPEED WRITE 0

(et appuyez sur ENTER) ou remettez la machine à zéro avec les touches CTRL, SHIFT et ESC si cela ne vous fait rien de perdre le programme en mémoire.

Chargement et exécution

Plus tard, quand vous commencerez à écrire vos propres programmes, vous voudrez savoir comment faire des programmes qui se chargent et s'exécutent automatiquement comme dans la bande de démonstration « Bienvenue ». La méthode est toute simple. Vous tapez la commande SAVE, suivie du nom du programme comme d'habitude. Vous faites

ensuite suivre cela de la lettre P, qui signifie « protégé ». Par exemple vous pourriez avoir une commande comme :

SAVE " longprof ", P

utilisée pour sauver votre programme. La procédure SAVE fonctionne comme d'habitude, mais ce qui est placé sur la bande n'est pas normal. Pour recharger ce programme vous devez soit :

1) Appuyer sur CTRL et la *petite* touche ENTER, puis appuyer sur la touche PLAY de l'enregistreur, suivie par n'importe quelle touche.

soit :

2) taper RUN, puis appuyer sur ENTER (la touche ENTER *principale*), et manipuler l'enregistreur de cassette comme auparavant.

De l'une ou l'autre manière, le programme se chargera et s'exécutera automatiquement. Mais vous *ne* pourrez *pas* le lister ou l'exécuter à l'aide de RUN une deuxième fois, après qu'il sera fini. Ne sauvez jamais un programme avec protection si vous n'avez pas d'autre copie; parce qu'un programme ainsi protégé sera perdu si quoi que ce soit se passe mal, et vous ne pourrez pas en faire la copie.

Les méthodes de protection aident un peu à réduire le piratage des programmes, mais elles punissent injustement l'utilisateur honnête qui ne sait pas comment faire des copies de sécurité de tels programmes. Vous pouvez toutefois vous attendre à trouver dans les revues des indications pour tourner ces méthodes de protection, quand vous serez plus avancé en programmation.

Quand vous êtes prêt

La première fois que vous lirez ce livre, vous pourrez sans dommage sauter ce qui suit, car cela ne vous sera utile que lorsque vous aurez plus d'expérience en programmation.

Il y a deux autres méthodes pour charger un programme, outre LOAD et RUN.

Vous pouvez utiliser CHAIN, suivi d'un nom de fichier. Cela chargera le programme et l'exécutera aussitôt, si bien que vous n'avez pas besoin d'utiliser LOAD et puis RUN.

Un autre type de chargement est offert par MERGE. Normalement quand vous chargez un programme, cela efface tout programme préexistant dans la mémoire. En utilisant MERGE, vous *ajouterez* un programme à un autre, de sorte que vous pouvez transformer toute une série de petits programmes en un grand. Les programmes doivent avoir des ensembles de numéros de lignes différents.

Vous pouvez utiliser CHAIN et MERGE ensemble (comme CHAIN MERGE "BITS) pour ajouter un programme à un autre et exécuter le programme résultant. Une autre facilité offerte est de lire le catalogue de tous les programmes d'une bande, en utilisant CAT. Cela est plus utile pour de courtes bandes enregistrées à vitesse rapide.

2

Tout mettre sur l'écran

Le chapitre 1 vous aura introduit à l'idée que le CPC464, comme presque tous les ordinateurs, prend ses ordres dans ce que vous tapez sur le clavier. Vous aurez aussi constaté qu'un ordre est exécuté lorsque la touche ENTER est enfoncée. Vous aurez déjà utilisé la commande LIST qui affiche les instructions de votre programme sur l'écran.

Vous devez connaître quelques autres points utiles avant d'aller plus loin. Le premier est que vous pouvez nettoyer l'écran en tapant CLS (ou cls) et puis en appuyant sur ENTER. Et comme votre familiarité avec le clavier d'ordinateur augmente, vous allez avoir besoin d'utiliser les commandes d'édition, qui sont expliquées dans l'*appendice A.*

Mode direct et mode programme

Il y a deux manières d'utiliser un ordinateur. L'une est appelée le *mode direct.* Mode direct signifie que vous frappez une commande, appuyez sur ENTER, et que la commande est exécutée immédiatement. Cela peut être utile, mais l'usage le plus important pour un ordinateur réside dans ce qu'on appelle le *mode programme.* En mode programme, l'ordinateur reçoit un ensemble d'instructions, avec un guide d'utilisation quant à l'ordre dans lequel les effectuer. Un ensemble d'instructions est appelé un *programme,* quand il est assorti de ce guide. La différence entre ces deux modes est importante, car les instructions d'un programme peuvent être répétées autant de fois que vous le voulez sans grand effort de votre part. Par contre, une commande directe ne sera répétée

que si vous la retapez entièrement, et réappuyez sur ENTER. L'ensemble des mots de commande et d'instructions qui peuvent être utilisés, et les règles qui président à leur usage forment ce qu'on appelle un *langage de programmation.* Le CPC464 vous offre une version nouvelle et très moderne du langage de programmation le plus utilisé sur les petits ordinateurs, le BASIC. BASIC est l'abréviation de « Beginners All-purpose Symbolic Instruction Code » (Code d'instructions symbolique à tous usages pour les débutants). Ce langage a d'abord été conçu pour l'enseignement de la programmation. La version de BASIC que votre CPC464 utilise est toutefois enrichie de beaucoup d'instructions supplémentaires.

Il est très important de respecter précisément la forme de tout terme du vocabulaire d'un ordinateur, qu'il s'agisse de commandes directes ou d'instructions de programme. Par exemple, l'*orthographe* d'un mot de commande doit être parfaite, sans cela l'ordinateur n'obéira pas. Il faut aussi *utiliser* les mots de la bonne manière. Par exemple vous ne pouvez pas juste taper SAVE et appuyer sur ENTER, et attendre que l'ordinateur fasse quoi que ce soit. L'ordinateur attend des guillemets derrière SAVE, et ne peut réagir à la commande si celle-ci n'est pas complète. Certaines commandes impliquent des *espaces* entre les mots, et ne fonctionneront pas si un espace est oublié. A d'autres endroits, vous constaterez que respecter les espaces entre mots n'a pas d'importance. Vous devez apprendre ces choses par expérience, car il n'y a pas de règle fiable quant à la répartition des endroits où il faut un espace, et ceux où l'espace est facultatif.

Le CPC464, comme vous le savez désormais, vous autorise à taper commandes et instructions en majuscules ou en minuscules, mais les traduit en majuscules lorsque le programme est listé sur l'écran. Comme cela peut être à l'origine de confusions quand vous essayez de suivre un listing imprimé, si les mots d'instructions sont tapés en minuscules, désormais, dans ce livre, toutes les instructions seront présentées, que ce soit dans le texte ou les listings, en majuscules. Vous savez que si vous les tapez en minuscules cela n'a pas d'importance, mais à l'intérieur du livre, adopter les majuscules pour les mots d'instructions ou de commandes rend les choses plus claires. Vous savez aussi qu'une commande (directe) doit être suivie d'un appui sur ENTER, aussi n'ai-je plus besoin de vous le rappeler en imprimant des phrases

comme « appuyer sur ENTER » à chaque fois que c'est nécessaire pour l'exécution d'une commande.

Voyons maintenant la différence entre une commande directe et une instruction de programme. Si vous voulez que l'ordinateur effectue la *commande directe* d'additionner deux nombres, 1.6 et 3.2, vous devez taper :

PRINT 1.6+3.2 (et appuyer sur ENTER)

(NB : comme la plupart des ordinateurs individuels, le CPC464 utilise la notation anglaise pour les nombres décimaux et donc un point y remplace la virgule.)

Vous devez commencer par PRINT (ou print), car un ordinateur est une machine inintelligente, qui n'obéit qu'à un petit ensemble d'instructions. Si vous n'utilisez pas le mot PRINT, l'ordinateur ne peut reconnaître ce que vous voulez, c'est-à-dire voir la réponse sur l'écran. Il ne reconnaît pas des instructions comme DONNEZ-MOI ou COMBIEN FONT, il connaît seulement quelques mots qu'on appelle *mots réservés* ou *mots d'instruction.* PRINT est l'un de ces mots. Souvenez-vous qu'il *doit* y avoir un espace derrière le T de PRINT. Vous n'avez pas besoin d'introduire d'espaces entre 6 et + ou entre + et 3.

Quand vous appuyez sur la touche ENTER après avoir frappé PRINT 1.6+3.2, l'écran montre la réponse, 4.8. Cette réponse n'est pas affichée à l'endroit où vous avez tapé la commande, mais à la ligne suivante, et elle commence décalée d'un espace par rapport à la marge de gauche de l'écran. Si vous venez de nettoyer l'écran avant de taper, la réponse apparaîtra près du coin supérieur gauche. Quand vous travaillez avec un moniteur couleur ou une TV, le CPC464 nettoie l'écran en utilisant un fond de couleur bleue.

Tant que nous en sommes à PRINT, il y a une particularité à signaler : vous pouvez taper un point d'interrogation (?) à la place de PRINT. Qui plus est, si vous utilisez le point d'interrogation, vous n'avez pas besoin de laisser d'espace entre lui et le premier chiffre. Vous pouvez taper :

?1.6+3.2

et appuyer sur ENTER pour voir le résultat sans craindre le redoutable message d'erreur de syntaxe. Puisque utiliser le

point d'interrogation évite autant de dactylographie, il est très utile de connaître cette astuce lorsque vous avez beaucoup de lignes qui commencent par une instruction PRINT.

Une fois qu'une commande directe a été effectuée, cependant, elle est achevée une fois pour toute. Un *programme* ne fonctionne pas de la même manière. Un programme est tapé, mais les instructions du programme ne sont pas effectuées quand vous appuyez sur la touche ENTER. A la place, les instructions sont stockées en mémoire, prêtes à être exécutées comme et quand vous le voulez. L'ordinateur doit avoir un moyen de reconnaître la différence entre vos commandes et vos instructions de programme. Sur les ordinateurs qui utilisent le « langage » appelé BASIC, cela se fait en commençant chaque instruction de programme par un nombre qu'on appelle *numéro de ligne.* Ce nombre doit être un *nombre entier positif.* C'est pourquoi vous ne pouvez espérer que l'ordinateur comprenne une instruction comme 5.6+3=, parce qu'il prend 5 comme un numéro de ligne, et le reste n'a aucun sens pour lui.

Premiers pas en programmation

Commençons donc à programmer, en utilisant les opérations arithmétiques d'addition, de soustraction, de multiplication et de division. Ce choix n'a d'autre motif que la simplicité de mise en œuvre. Les ordinateurs ne sont pas seulement utilisés pour faire des calculs, mais il est utile de savoir en faire à l'occasion. La *figure 2.1* montre un programme de quatre lignes qui imprimera des résultats arithmétiques.

Regardez-le attentivement, car il y a beaucoup à apprendre dans ces quatre lignes. Pour commencer, les numéros de lignes sont 10, 20, 30, 40 plutôt que 1,2,3,4. C'est pour laisser

```
10 PRINT 5.6+6.8
20 PRINT 9.2-4.7
30 PRINT 3.3*3.9
40 PRINT 7.6/1.4
```

Fig. 2.1. Un programme arithmétique de quatre lignes.

des possibilités de modification, en cas de remords. Si vous décidez, en effet, que vous voulez une autre instruction entre les lignes 10 et 20, vous pouvez taper le numéro de ligne 15, ou 11 ou 12 ou n'importe quel nombre entier entre 10 et 20, et le faire suivre de votre nouvelle instruction. Même si vous avez entré la ligne après la 30 par exemple, l'ordinateur la placera automatiquement dans l'ordre entre les lignes 10 et 20. Si vous numérotez les lignes 1,2,3, il n'y a pas d'espace pour ces remords (quoique vous puissiez changer les numéros de lignes, si nécessaire, en utilisant les commandes d'édition).

Il faut ensuite remarquer comment le chiffre zéro est barré sur l'écran. C'est pour le distinguer de la lettre O. L'ordinateur refusera simplement le 0 à la place du O, et vice versa. La barre rend la différence plus évidente, de sorte que vous êtes moins à même de faire des erreurs. Le zéro que vous voyez sur le clavier est aussi barré, il se trouve sur une touche différente, et a une forme différente. Frappez des zéros et des O sur l'écran de manière à voir la différence. Sur les listings vous verrez le zéro barré, mais il ne le sera pas dans le texte.

Maintenant des points plus importants. L'étoile ou astérisque à la ligne 30 est le symbole que le CPC464 utilise comme *signe multinlier.* Une fois de plus, on ne peut utiliser ×, que vous pourriez normalement utiliser pour indiquer la multiplication, pour éviter la confusion avec x. Il n'y a pas non plus de *signe diviser* sur le clavier, aussi le CPC464 utilise-t-il comme tous les autres petits ordinateurs le signe barre oblique (/) à la place. C'est la ligne en diagonale qui se trouve sur la même touche que le point d'interrogation, *pas* celle qui se trouve sur la touche juste à côté.

Jusqu'ici, tout va bien. On entre le programme en le frappant juste comme vous le voyez imprimé. Vous n'avez pas besoin de laisser d'espace entre le numéro de ligne et le P de PRINT, car le CPC464 en mettra un pour vous quand il affichera le programme sur l'écran. Vous *devez,* en revanche, laisser un espace après PRINT, et je dois attirer votre attention sur ce point, parce que tous les ordinateurs ne sont pas aussi tâtillons que celui-ci à ce propos. Si vous utilisez l'abréviation ? pour PRINT vous n'avez pas à vous soucier de l'espace.

Pour en revenir à l'exemple du programme, vous aurez à appuyer sur ENTER à chaque fois que vous aurez achevé une

ligne d'instruction, avant de taper le numéro de ligne suivant. Vous devriez aboutir à un programme comme celui de l'illustration. Une fois que vous avez entré le programme en le dactylographiant, il se trouve stocké dans la mémoire de l'ordinateur sous la forme d'un ensemble de nombres-codes. Il y a deux choses que vous devez savoir maintenant. L'une est la manière de vérifier que le programme est effectivement en mémoire, l'autre comment faire en sorte que la machine exécute les instructions du programme.

La première partie est réglée par l'utilisation de la commande LIST que vous connaissez déjà. Vous pouvez utiliser la touche CLS pour effacer l'écran auparavant si vous le souhaitez, puis taper LIST et appuyer sur la touche ENTER. C'est seulement quand vous aurez appuyé sur ENTER, et pas avant, que votre programme sera listé sur l'écran. Vous verrez alors comment l'ordinateur a affiché les items de programme sur l'écran, avec des espaces entre les numéros de lignes et les instructions. Les signes ? ont été convertis en mots PRINT, et un espace a été inséré entre le mot et le premier chiffre du nombre.

Pour mettre en œuvre le programme, il vous faut une autre commande, RUN. Tapez RUN, puis appuyez sur ENTER, et vous verrez l'exécution des instructions. Pour être plus précis, vous verrez :

```
12.4
4.5
12.87
5.42857143
```

Cette dernière ligne devrait vous donner une idée de la précision que le CPC464 peut apporter à ce genre de calcul. Vous remarquerez d'ailleurs que, si vous avez listé le programme avant de le faire exécuter par RUN, les résultats sont affichés sous le listing du programme. Vous pouvez éviter cela en utilisant CLS (puis appuyez sur ENTER) avant d'utiliser RUN.

Quand vous faites suivre l'instruction PRINT d'une opération arithmétique comme 2.8*4.4, ce qui est affiché à l'exécution du programme est le *résultat* de l'opération. Le programme *n'*affiche *pas* 2.8*4.4, il affiche juste le résultat de l'opération 2.8*4.4.

Affichage et impression

Bien que cela puisse être utile, il n'est pas toujours commode d'avoir un ensemble de réponses sur l'écran, surtout si vous avez oublié quelles étaient les questions. Le CPC464 vous permet d'afficher tout ce que vous voulez sur l'écran, exactement comme vous le tapez, en utilisant ce qu'on appelle une *chaîne.*

```
10 PRINT "2+2="2+2
20 PRINT "2.5*3.5="2.5*3.5
30 PRINT "9.4-2.2="9.4-2.2
40 PRINT "27.6/2.2="27.6/2.2
```

Fig. 2.2. Usage des guillemets. Dans cet exemple et dans d'autres, l'abréviation ? a été utilisée à la place de l'instruction PRINT, mais PRINT apparaît sur le listing.

La *figure 2.2* illustre ce principe. A chaque ligne, une partie du texte tapé est incluse entre des guillemets, tandis que le reste ne l'est pas. Entrez le court programme de la *figure 2.2,* nettoyez l'écran et exécutez-le. Voyez-vous comme l'ordinateur a différemment traité les instructions ? Tout ce qui était inclus entre les guillemets a été affiché *exactement* comme vous l'avez tapé. Tout ce qui était à l'extérieur des guillemets a été effectué, si bien que la première ligne, par exemple, donne le résultat sans surprise :

2+2= 4

Maintenant cela n'a rien d'automatique. Si vous tapez une nouvelle ligne :

15 PRINT "2+2="5*1.5

vous obtiendrez, à l'exécution, la réponse hardie :

2+2= 7.5

L'ordinateur fait ce qu'on lui dit, et c'est ce que vous lui avez dit de faire. Seul un lunatique croirait que les ordinateurs peuvent prendre le contrôle du monde !

C'est le bon moment pour remarquer quelque chose d'autre. La ligne 15 que vous avez ajoutée a été insérée en place, entre

les lignes 10 et 20 — LISTez si vous ne le croyez pas. Peu importe dans quel ordre vous tapez les lignes de votre programme, l'ordinateur les triera en ordre croissant des numéros de lignes pour vous. De plus vous verrez que l'ordinateur a mis un espace entre le signe = et le premier chiffre de la réponse. C'est pour permettre l'introduction d'un signe + ou —, et il peut arriver que nous voulions un espace de plus à cet endroit. Cela peut se faire en laissant un espace entre le signe = et les guillemets finals de la ligne (comme = ").

Avec toute cette accumulation de savoir derrière nous, nous pouvons commencer à examiner d'autres formes d'affichage. PRINT, utilisé seul de cette manière, signifie toujours afficher sur l'écran TV. Pour mettre en marche une imprimante à papier (ce qu'on appelle une *sortie imprimée* ou hard copy), il y a une variété de l'instruction PRINT qui est suivie d'un signe distinctif (#) et du nombre 8. PRINT#8, par exemple, enverra l'impression vers une imprimante, *si vous en avez une de branchée.* Si vous n'avez pas d'imprimante branchée, vous devez éviter cette commande, parce qu'elle rendra l'ordinateur comme verrouillé, sans réaction aux touches, et inactif. Si vous vous retrouvez dans cette situation, vous avez la solution de brancher une imprimante, ou simplement d'appuyer sur ESC deux fois pour vous tirer d'embarras.

L'*appendice B* illustre comment utiliser une imprimante parallèle, telle l'Epson RX80, avec le CPC464.

```
10 PRINT "Voici"
20 PRINT "l'excellent"
30 PRINT "ordinateur Amstrad CPC464"
```

Fig. 2.3. Usage de l'instruction PRINT pour placer des mots sur l'écran.

Maintenant essayez le programme de la *figure 2.3.* Vous pouvez essayer de taper les lignes dans l'ordre que vous voulez, pour vérifier qu'elles seront rangées en ordre croissant des numéros de ligne quand vous listerez le programme. Quand vous effacez l'écran et exécutez à l'aide de RUN le programme, les mots apparaissent sur trois lignes séparées.

C'est parce que l'instruction PRINT ne signifie pas uniquement afficher sur l'écran. Elle signifie également passez à la ligne et commencez à gauche ! Vous constaterez aussi, incidemment, que quand les mots atteignent la ligne inférieure de l'écran, toutes les lignes semblent monter, et la ligne supérieure disparaît. C'est ce qu'on appelle le *déroulement* (mode rouleau ou scrolling), et ainsi la machine peut afficher un grand nombre de lignes sur un écran qui n'en contient que 25 en tout.

Maintenant, sélectionner une nouvelle ligne pour l'exécution de chaque instruction PRINT n'est pas toujours commode, et nous pouvons modifier ce comportement en utilisant des signes de ponctuation qu'on appelle modificateurs d'impression.

Prenez maintenant une nouvelle habitude en commençant ce programme : tapez NEW et appuyez sur la touche ENTER. Cela efface le vieux programme, et vous pourriez aussi souhaiter utiliser l'effet de CLS pour effacer l'écran. Si vous n'utilisez pas l'effet de NEW, vous risquez de trouver des lignes du vieux programme au milieu des nouvelles. Chaque fois que vous tapez une ligne, vous détruisez toute ligne de même numéro provenant d'un ancien programme, mais s'il y a un numéro de ligne que vous n'utilisez pas dans le nouveau programme, cette dernière ligne restera en mémoire. Dans la *figure 2.2*, par exemple, la ligne 15 que vous avez ajoutée resterait en mémoire même si vous aviez tapé une nouvelle ligne 10 et une nouvelle ligne 20.

```
10 PRINT "Voici ";
20 PRINT "l'excellent ";
30 PRINT "Amstrad CPC464"
```

Fig. 2.4. L'effet des points-virgules.

Maintenant essayez le programme de la *figure 2.4.* Il y a une différence très importante entre la *figure 2.4* et la *figure 2.3,* comme vous allez le voir quand vous ferez exécuter le programme à l'aide de RUN. L'effet du *point-virgule* suivant les derniers guillemets dans une ligne est d'empêcher l'affichage

suivant de commencer à la ligne suivante à gauche. Quand vous exécutez ce programme, tous les mots apparaissent sur une seule ligne. Il aurait été beaucoup plus simple d'avoir une seule ligne de programme disant :

10 PRINT "Voici l'excellent Amstrad CPC464"

pour obtenir le même résultat, mais il y a des fois où vous êtes obligé d'utiliser le point-virgule pour contraindre deux items d'affichage à paraître sur la même ligne. Nous verrons ce genre de cas plus tard, dans des exemples de programmes. Entre-temps, essayons une petite modification. Changez la ligne 30 de manière à ce qu'elle dise :

30 PRINT "ordinateur Amstrad CPC464"

et exécutez le programme à nouveau, à l'aide de RUN. Cette fois vous verrez un résultat tout différent. Les deux premières lignes ont été liées, mais la dernière ligne est indépendante. C'est parce que si la dernière ligne avait été liée à la précédente, un mot aurait été coupé entre deux lignes (car une ligne ici ne peut compter que 40 caractères ou espaces). Le CPC464 ne l'autorise pas, et c'est un trait unique de cette machine, qui rend plus facile d'organiser un affichage plus propre.

Lignes et colonnes

Un affichage propre dépend de l'organisation de vos mots et nombres en lignes et colonnes, aussi allons-nous maintenant examiner de plus près cette technique. Pour commencer, nous savons déjà que l'instruction PRINT entraîne le passage à une nouvelle ligne, aussi l'effet du programme de la *figure 2.5* ne devrait pas trop vous surprendre. Les lignes 10 et 20 contiennent une nouveauté, toutefois, sous la forme de deux instructions sur la même ligne. Les instructions sont séparées par *deux points* (:), et vous pouvez, si vous le voulez, mettre de cette manière, plusieurs instructions à la suite sur la même ligne, en prenant plusieurs lignes d'écran. La seule limite pratique à ce procédé est que cela rend vos instructions trop difficiles à lire si vous en entassez beaucoup ainsi.

Dans une ligne à instructions multiples de ce type, le CPC464 traite les différentes instructions de gauche à droite. L'instruction CLS ne devrait pas vous surprendre non plus — elle

nettoie l'écran, et fait commencer l'affichage au coin supérieur gauche. C'est le même effet que celui de la commande directe CLS, mais exécuté automatiquement à l'intérieur du programme.

```
10 CLS:PRINT "Voici le CPC464"
20 PRINT:PRINT
30 PRINT "Pret à travailler pour vous."
```

Fig. 2.5. Nettoyage de l'écran avec l'instruction CLS, et usage de lignes à plusieurs instructions. L'effet de la touche CLS n'est pas le même que celui de CLS dans une ligne de programme.

Un autre point à noter sur la *figure 2.5* est que la ligne 20 a pour effet de séparer les lignes. Les deux instructions PRINT, sans rien à afficher, entraînent chacune l'insertion d'une ligne vide. Il y a d'autres moyens pour cela, comme nous le verrons, mais celui-ci est très commode comme méthode pour créer un espace.

```
10 PRINT 1,2
20 PRINT 1,2,3,4
30 PRINT "UN","DEUX"
40 PRINT "UN","DEUX","TROIS","QUATRE"
50 PRINT "CET ITEM EST PLUS LONG","DEUX"
60 PRINT 1,2,3,4,5,6
```

Fig. 2.6. Comment la virgule répartit les mots en colonnes.

La *figure 2.6* traite des *colonnes.* La ligne 10 est une instruction PRINT qui agit sur les nombres 1 et 2. Quand ceux-ci sont affichés sur l'écran, ils apparaissent espacés, tout comme si l'écran avait été divisé en colonnes. Le signal qui cause cet effet est la *virgule* (,), et son action est complètement automatique. La virgule est sur la touche voisine de la lettre M, et si vous utilisez l'apostrophe sur la touche 7, vous n'obtiendrez pas le même effet ! Les deux se ressemblent plutôt sur le clavier, mais sont complètement différentes sur l'écran. Comme la ligne 20 le montre, vous ne pouvez avoir

que trois colonnes, dont chacune comporte jusqu'à quatorze caractères, selon que vous utilisez des lettres ou des nombres. Tout ce que vous tenterez de mettre dans une quatrième colonne apparaîtra en fait à la première colonne de la ligne en dessous. L'effet est le même pour les mots et pour les chiffres, comme en témoignent les lignes 30 et 40. Toutefois, quand vous affichez des mots de cette manière, vous devez vous rappeler que les virgules doivent être en dehors des guillemets. Toute virgule placée à l'intérieur des guillemets sera affichée telle quelle, et ne produira aucun effet d'espacement. Vous constaterez aussi que si vous tentez de faire entrer dans une colonne quelque chose de trop long pour elle, la longue expression débordera sur la colonne suivante, et l'item à afficher suivant se trouvera au début de la colonne suivante. La ligne 50 illustre ce fait — la première expression déborde de la colonne 1 sur la colonne 2, et le mot DEUX est affiché au début de la colonne 3 sur la même ligne. La ligne 60 montre ce qui arrive si vous utilisez plus de virgules — les colonnes prennent les mêmes positions sur la ligne suivante.

Cet effet des virgules se retrouve sur pratiquement tous les ordinateurs familiaux, mais le CPC464 y ajoute un nouveau tour. Tapez ZONE 6, appuyez sur ENTER, et exécutez à nouveau votre programme à l'aide de RUN. Cette fois les choses vont mieux ! C'est comme si vous aviez soudain plus de colonnes, c'est d'ailleurs le cas, en fait ! Le nombre qui suit ZONE donne le nombre d'espaces entre les colonnes, aussi vous permet-il de marquer (invisiblement) l'écran pour afficher de la manière que vous voulez. Vous pouvez changer la valeur de ZONE durant un programme, par exemple, de sorte que le mode d'affichage soit diversifié. C'est particulièrement utile si vous voulez travailler avec des tabulations pour des projets professionnels.

Les virgules sont utiles quand vous cherchez un moyen simple de créer des colonnes également espacées. Il existe une méthode bien plus souple pour placer à volonté des mots sur l'écran. On programme cela à l'aide de l'instruction TAB, qui doit suivre PRINT. TAB est l'abréviation de « tabuler », et il signifie « diviser en colonnes ».

Pour utiliser TAB, nous devons nous rappeler que l'écran, dans sa configuration à l'allumage de l'ordinateur, est divisé

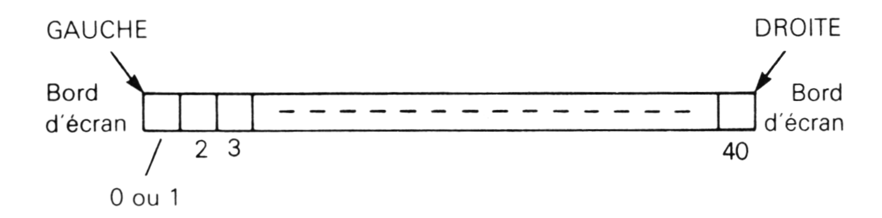


Fig. 2.7. Le schéma de TAB, qui montre la correspondance entre les nombres de TAB et les positions sur la ligne.

en 40 colonnes sur 25 lignes de haut. La *figure 2.7* montre un plan de ces numéros de colonnes pour TAB. Les positions sont numérotées de 1 à 40, mais vous pouvez aussi utiliser TAB(0), qui donne la même position que TAB(1), le bord gauche. TAB(1), alors, signifierait le bord gauche, et TAB(40) le bord droit. Le mot TAB doit suivre PRINT, et doit lui-même être suivi par le nombre de TAB entre parenthèses. Si vous oubliez les parenthèses, vous verrez bizarrement l'affichage d'un zéro avant votre nombre ou votre expression — vous comprendrez pourquoi plus tard.

```
10 PRINT TAB(18) "Centre"
20 PRINT TAB(5) "début ici"
30 PRINT TAB(45) "45 est ici"
```

Fig. 2.8. Comment TAB s'utilise pour placer le curseur dans un programme.

Maintenant essayez un exemple de TAB, comme dans la *figure 2.8.* Le premier mot est inscrit au centre de l'écran, et le second près du côté gauche. Le troisième mot, cependant, est affiché sur la ligne à la même position que le second. C'est parce que l'instruction TAB fonctionne avec des nombres de 1 à 40, et que si vous utilisez 41, cela revient à recommencer à 1. Vous *ne* passez *pas* à la ligne suivante en utilisant un nombre de TAB supérieur à 40, contrairement à ce qui se passe sur certaines machines.

Il y a un autre point à voir ici aussi. Le mot « centre » dans cet exemple a été affiché au centre de l'écran grâce à TAB (18). La *figure 2.9* montre la formule qui permet de trou-

1. Compter le nombre de caractères dans le titre, y compris les espaces.
2. Soustraire ce nombre de 40 s'il est pair, de 41 s'il est impair.
3. Diviser le reste par 2.
4. Utiliser le résultat comme nombre de TAB.

Fig. 2.9. La formule pour centrer un titre.

ver la valeur TAB pour centrer tout mot ou expression sur l'écran. Souvenez-vous cependant, quand vous l'utilisez, que l'écran du CPC464 n'est pas toujours parfaitement centré sur l'écran du moniteur ou de la TV. Sur mon moniteur couleur, par exemple, la marge de gauche est plus grande que celle de droite.

```
10 PRINT TAB(2) "Ordinateur"TAB(16)"CPC4
64"
20 PRINT TAB(2) "Ordinateur"SPC(16)"CPC4
64"
```

Fig. 2.10. Utilisation de SPC pour espacer l'affichage.

La *figure 2.10* présente un moyen assez différent d'espacer des chiffres ou des lettres sur l'écran. Elle fait usage de l'instruction SPC, et bien que vous puissiez croire qu'elle est toute semblable à l'instruction TAB, ce n'est pas le cas ! La *figure 2.10* vous montre la différence. Quand vous placez les mots à l'aide de TAB, la première lettre de chacun se trouve à la position de TAB appropriée. Notez incidemment que vous pouvez avoir plus d'un TAB sur une ligne après un PRINT. Contrairement à la plupart des ordinateurs, le CPC464 n'exige pas de point-virgule pour séparer les éléments de cette instruction.

A la ligne 20, SPC est utilisé. Or cela n'envoie pas la tabulation de position 16 — cela affiche 16 espaces entre la fin d'ORDINATEUR et le début de CPC464. C'est tout à fait différent d'avoir le C de CPC464 à la colonne 16. La règle générale est que vous devrez utiliser TAB si vous voulez des colonnes alignées avec la première lettre de chaque mot à la même

position sur chaque ligne. Vous utiliserez SPC si vous voulez fixer la quantité d'espaces entre mots ou nombres, même si ces mots ou nombres sont de différentes longueurs.

Il y a encore un autre moyen d'organiser votre affichage, et il est encore plus souple que TAB. L'instruction est LOCATE, qui vous permet de placer des mots à n'importe quelle position sur l'écran, et cela dans n'importe quel ordre. Normalement, quand vous affichez par PRINT sur l'écran, l'instruction PRINT force l'ordinateur à passer à la ligne, et vous ne pouvez revenir à la ligne précédente. En utilisant LOCATE, vous pouvez placer des mots et des nombres où il vous plaît, et vous pouvez même remplacer une ligne par une autre si vous le désirez.

La différence importante est que l'instruction LOCATE *doit* être énoncée avant l'instruction PRINT à laquelle elle se réfère. Elle peut se trouver sur la ligne juste avant l'instruction PRINT, ou sur la même ligne, séparée par deux points.

LOCATE doit être suivi par deux nombres, deux « paramètres ».

Le premier des deux est un *numéro de colonne.* Vous pouvez afficher 40 caractères (lettres, nombres, signes de ponctuation) sur une ligne de l'écran du CPC464. Chacun de ces caractères se trouve dans une colonne, parce que les caractères de toutes les autres lignes sont aussi répartis de la même manière. On peut donc utiliser un nombre pour représenter la position d'un caractère sur une ligne. Ce nombre peut aller de 1 (limite gauche) à 40 (limite droite), tout comme les valeurs de TAB.

Le deuxième paramètre de l'instruction LOCATE est un *numéro de ligne-écran.* Les lignes sur l'écran sont numérotées de 1 (ligne supérieure) à 25 (ligne inférieure de la partie utile à l'affichage de l'écran).

Un exemple serait certainement utile ici. Jetez un coup d'œil à la *figure 2.11.* La ligne 10 nettoie l'écran, et la ligne 20 introduit le premier LOCATE. Les valeurs sont 1,1, ce qui signifie que l'affichage va commencer au coin supérieur gauche de l'écran. Il *doit* y avoir un espace entre le E de LOCATE et le premier nombre. Le S de SUPÉRIEUR GAUCHE se trouvera donc au coin supérieur gauche de l'écran, comme il se trouverait d'ailleurs de toute manière juste après CLS. La ligne sui-

```
10 CLS
20 LOCATE 1,1:PRINT "SUPERIEUR GAUCHE"
30 FOR N=1 TO 1000:NEXT
40 LOCATE 26,25:PRINT "INFERIEUR DROIT";
50 FOR N=1 TO 1000:NEXT
60 LOCATE 1,25:PRINT "INFERIEUR GAUCHE"
70 FOR N=1 TO 1000:NEXT
80 LOCATE 26,1:PRINT "SUPERIEUR DROIT";
90 FOR N=1 TO 1000:NEXT
100 LOCATE 17,12:PRINT "MILIEU"
```

Fig. 2.11. Comment LOCATE vous permet d'afficher n'importe où sur l'écran.

vante cause une pause. Nous n'avons pas encore vu ce genre d'instruction, mais nous allons y venir plus tard.

Le LOCATE suivant a pour valeurs 26,25, de sorte que l'expression INFÉRIEUR DROIT soit placée avec le T de DROIT au coin inférieur droit de l'écran. Comment cela a-t-il été calculé ? En comptant le T de DROIT comme colonne 40, puis en décomptant 39, 38, 37 et ainsi de suite jusqu'au I, ce qui faisait 26. Enfin on est à la ligne du bas, c'est-à-dire la ligne 25. Pour empêcher l'écran de se dérouler à ce point, j'ai dû ajouter le point-virgule qui suit l'expression. Normalement, le point-virgule n'est pas nécessaire, mais le T d'INFÉRIEUR DROIT vient à la dernière position de l'écran, et ce fait déclencherait normalement le déroulement. Avec ces informations, vous pouvez voir comment le reste des mots a été mis en place. Grâce aux pauses, vous pouvez constater que nous ne suivons pas l'ordre normal pour l'affichage de gauche à droite et de haut en bas.

Mise en page d'un écran

La *figure 2.12* est un schéma que vous trouverez utile pour placer les mots où vous le voulez sur l'écran à l'aide de LOCATE. Cette grille montre plusieurs ensembles de coordonnées, et il se peut que vous vous en demandiez la raison. C'est parce que vous pouvez choisir trois tailles différentes de caractères sur votre écran.

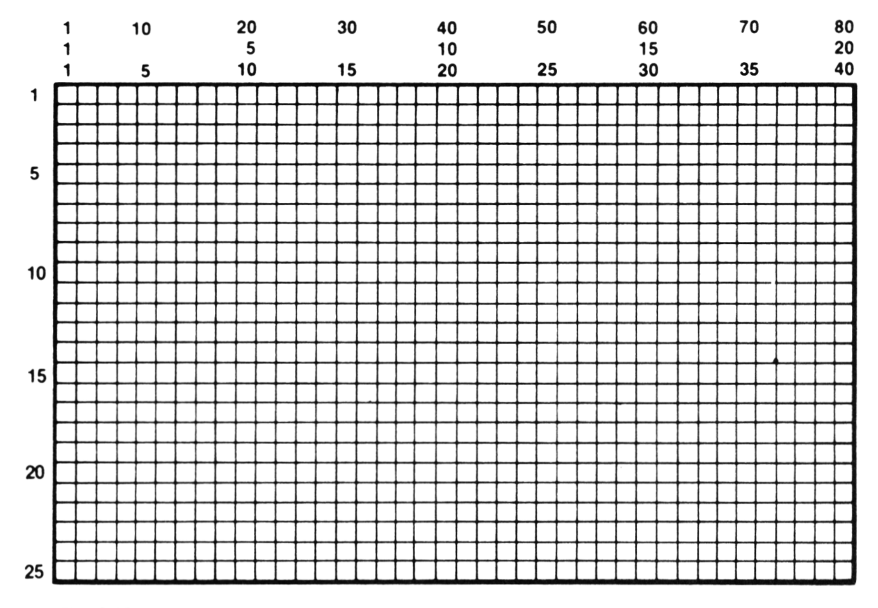


Fig. 2.12. Une matrice pour faire le meilleur usage de LOCATE.

Conservez en mémoire le programme de la *figure 2.11,* et tapez MODE 0 (puis ENTER). Il *doit* y avoir un espace entre le E de MODE et le zéro. Maintenant si vous listez votre programme, vous allez constater que tous les caractères ont une taille double de celle qu'ils avaient précédemment. Quand vous exécuterez le programme, vous verrez que le « supérieur gauche » et l' « inférieur gauche » sont correctement placés, mais pas les messages de droite. C'est parce que les valeurs de LOCATE= vont désormais de 1 à 20, en colonnes, bien que l'intervalle reste de 1 à 25 en lignes.

Essayez maintenant de taper MODE 2 (puis ENTER). Cette fois, vous constaterez que les caractères sont beaucoup plus petits, il y en a 80 par ligne. Ils sont aussi beaucoup plus difficiles à lire, surtout sur un moniteur couleur ou un téléviseur. Les valeurs horizontales de LOCATE vont maintenant de 1 à 80. Quand vous allumez le CPC464, le MODE est toujours le MODE 1, avec 40 caractères par ligne. C'est un excellent compromis entre le confort de lecture des caractères et le nombre raisonnable de caractères présents sur l'écran. A moins que vous n'utilisiez le moniteur à écran vert, il est préférable d'éviter les lettres du mode 2.

Numérotation automatique et renumérotation

Un dernier point. Le CPC464 possède deux des commandes qui manquent à tant de machines.

L'une est AUTO. Si vous tapez AUTO (puis ENTER), la machine numérotera automatiquement les lignes pour vous. Tout ce que vous avez à faire est de taper ce qui doit apparaître sur chaque ligne. Chaque fois que vous appuyez sur ENTER, un autre numéro de ligne apparaîtra pour vous ! Pour arrêter l'auto-numérotation, appuyez sur ESC deux fois. Vous pouvez aussi faire commencer l'action d'AUTO à un numéro à votre gré (essayez AUTO 100, par exemple), et vous n'êtes pas non plus obligé d'avoir des lignes incrémentées de 10 en 10 (essayez AUTO 10,5). Cette fonction est très commode si vous recopiez un programme à partir d'un listing imprimé, et que le listing ait été renuméroté de dix en dix.

RENUM est l'autre commande. Tapez RENUM, et votre programme se renumérotera à partir de 10, avec des lignes incrémentées par 10, quel que soit le nombre de lignes intercalées. Vous pouvez choisir de faire partir votre renumérotation d'une ligne existante, disons 240, vous pouvez choisir le numéro que vous souhaitez pour la première ligne (une fois la renumérotation effectuée) et vous pouvez choisir le pas d'incrémentation (la quantité ajoutée à chaque numéro de ligne). Croyez-le ou non, il y a des machines qui n'offrent pas ces facilités !

3

Quelques variations

Jusqu'ici notre utilisation de l'ordinateur s'est limitée à l'affichage de nombres et de mots sur l'écran. C'est l'un des principaux objectifs de l'informatique, mais nous devons examiner maintenant quelques aspects de ce qui se passe avant que quoi que ce soit ne s'affiche. L'un des points à reconnaître est appelé l'*assignation.*

```
10 CLS
20 X=23
30 PRINT "2 FOIS"X" FONT";2*X
40 X=5
50 PRINT "X VAUT MAINTENANT"X
60 PRINT "ET LE DOUBLE DE"X" EST";2*X
```

Fig. 3.1. L'effet de l'assignation. La lettre X a été utilisée à la place d'un nombre.

Jetez un coup d'œil au programme de la *figure 3.1.* Tapez-le, exécutez-le et comparez ce que vous voyez sur l'écran avec ce qui apparaît dans le programme. La première ligne d'affichage est la ligne 30. Ce qui apparaît sur l'écran est :

2 FOIS 23 FONT 46

mais les nombres 23 et 46 n'apparaissent pas à la ligne 30 ! C'est parce que nous avons utilisé la lettre X comme une sorte de code pour le nombre 23. L'appellation officielle de ce type de code est *identificateur de variable.*

La ligne 20 assigne l'identificateur de variable X (on dit sou-

vent simplement la variable), en lui donnant la valeur de 23. « Assigne » signifie que chaque fois que nous utiliserons X *à l'extérieur* de guillemets, l'ordinateur travaillera avec le nombre 23 à la place. Comme X est un seul caractère et que 23 a deux chiffres, on gagne de l'espace. Le gain aurait été encore plus grand si nous avions assigné X différemment, avec quelque chose comme X=2174.3256 par exemple. La ligne 30 prouve ensuite que X est pris pour 23, parce que chaque apparition de X hors des guillemets donne lieu à l'affichage de 23, et l'expression 2*X s'affiche 46. Toutefois, X n'a pas définitivement pour valeur 23.

La ligne 40 assigne 5 à X, et les lignes 50 et 60 prouvent que le changement a été effectué.

C'est pourquoi on appelle X une *variable* : on peut faire varier ce que l'on veut que X représente. Mais X reste assigné tant qu'on ne le change pas. Même après la fin de l'exécution du programme de la *figure 3.1,* pourvu que vous n'ayez ni ajouté ni supprimé de lignes, vous pouvez taper PRINT X, et le fait d'appuyer sur ENTER fera apparaître la valeur de X sur l'écran.

Le listing montre aussi une curiosité : vous pouvez faire suivre une expression entre guillemets par un identificateur de variable, comme dans PRINT "2 FOIS"X, à la ligne 30, sans point-virgule. Mais vous *ne* pouvez *pas* en faire autant si l'expression est suivie par une formule à calculer comme 2*X ou X—7. Vous obtiendrez un message d'erreur si vous essayez d'utiliser dans le programme "FONT"2*X. Aussi beaucoup de programmeurs utilisent-ils automatiquement un point-virgule chaque fois que des guillemets sont suivis par un identificateur de variable ou une expression à calculer.

Pour en revenir aux variables, elles fournissent un moyen très commode de manipuler des nombres sous forme codée. L'identificateur qui remplace le nombre doit commencer par une lettre, majuscule ou minuscule. Vous pouvez ajouter à cette première lettre d'autres lettres, pour faire un mot complet si vous le souhaitez, ou des chiffres, mais pas d'espaces ou de signes arithmétiques (+—*/), ni de signes de ponctuation d'aucune sorte.

Des identificateurs comme TOTAL, nom, TOUTCEQUILYA et R2D2 peuvent tous être utilisés comme identificateurs de

variables numériques, et chacun peut recevoir un nombre différent. Vous devez toutefois vous rappeler que *le CPC464 ne distingue pas les identificateurs en majuscules et en minuscules.* Si vous assignez l'identificateur de variable conf au nombre 45, puis assignez CONF à 88, vous constaterez que conf et CONF ont tous deux la même valeur, 88 dans ce cas. Cette valeur sera la plus récente assignée. Il faut aussi veiller aux *mots réservés.* Les mots réservés du CPC464 sont ses mots d'instructions — des mots comme PRINT, NEW, RUN, etc. Vous *ne* pouvez *pas* les utiliser comme identificateurs de variable, d'ailleurs vous obtiendrez un message d'erreur si vous tentez de le faire.

Certains ordinateurs vous interdiront même d'utiliser des mots qui *contiennent* ces mots réservés, ce qui vous empêcherait d'utiliser comme identificateur de variable RUNES par exemple. Mais le CPC464 est plus tolérant, et vous permettra d'employer tout mot qui n'est pas absolument identique à un mot réservé. Si vous demandez au CPC464 de vous afficher la valeur d'un identificateur qui n'a pas reçu d'assignation, il répondra avec la valeur 0, et non avec un message d'erreur, comme le font certaines machines.

Pour augmenter le bénéfice de cette facilité, vous pouvez utiliser des identificateurs similaires pour représenter des mots ou des expressions. La différence est que vous devez ajouter un *signe dollar* ($) à l'identificateur de variable. Si N est une variable numérique, N$ (qu'on lit n-dollar — le s barré de dollar signifie en anglais “string”, chaîne de caractères) est un identificateur de mot ou d'expression.

L'ordinateur traite ces deux variables N et N$ comme des objets parfaitement séparés et différents. Leur assignation se passe d'ailleurs de manière assez différente. Quand vous assignez un nombre à une variable numérique en utilisant le signe =, vous n'avez pas besoin de taper de guillemets de part et d'autre du nombre. Quand vous assignez une chaîne, également à l'aide du signe =, au contraire, vous *devez* faire usage des guillemets. Nous verrons ultérieurement d'autres méthodes d'assignation. Dans l'exemple suivant, nous utiliserons des identificateurs de variables longs, c'est-à-dire de plus de deux lettres. Utiliser une seule lettre économise de l'espace, mais un bon choix de noms de variables peut vous rappeler efficacement ce que la variable représente. Quand

vous avez autant de mémoire disponible qu'en propose le CPC464, vous pouvez vous permettre d'être généreux avec vos identificateurs de variables.

Variables chaînes

```
10 CLS:NOM$=" CPC464"
20 PREM$="L'excellent":DER$ =" ordinateu
r"
30 PRINT PREM$;" ";DER$;" ";NOM$
40 PRINT "Ceci utilise le ";NOM$
50 PRINT PREM$;" ";NOM$;" en action!"
```

Fig. 3.2. L'usage des variables chaînes. Elles sont signalées par le signe dollar.

La *figure 3.2* illustre les *variables chaînes,* c'est-à-dire l'utilisation d'identificateurs de variables pour des mots ou des expressions — des « chaînes de caractères ». Les lignes 10 et 20 effectuent les opérations d'assignation, et les lignes 30 à 50 montrent comment ces identificateurs de variables peuvent être utilisés. Notez que vous pouvez utiliser un identificateur de variable, qui n'a pas besoin de guillemets à côté de texte ordinaire, qui lui *doit* être encadré de guillemets. Vous devez être soigneux quand vous mêlez les deux, car il est facile de mélanger les mots. Remarquez sur les lignes 30 à 50 comment on a ménagé les espaces entre les mots. Quand on affiche une variable après une autre, l'espace est créé en tapant des guillemets, en appuyant sur la barre d'espacement et en fermant les guillemets : " ". Pour laisser un espace entre du texte entre guillemets et une variable, il suffit d'appuyer sur la barre d'espacement avant de fermer les guillemets, à l'endroit où vous voulez le blanc. Les points-virgules *ne* sont *pas essentiels* quand vous combinez des éléments de texte de cette manière. Si vous omettez le point-virgule à une jonction, vous obtiendrez le même résultat que si vous l'aviez inclus, pourvu que vous n'utilisiez pas d'expression numérique à calculer.

La *figure 3.3* montre un autre exemple, utilisant cette fois les identificateurs de variables TRUC$ et SUPER$ pour des expressions plus longues. Il n'y aurait aucune raison de pro-

```
10 CLS:TRUC$="Le nouvel ordinateur."
20 SUPER$="De la vraie informatique pour
 moins cher"
30 PRINT"Le CPC464- "
40 PRINT TRUC$;SUPER$
50 PRINT TRUC$+SUPER$
```

Fig. 3.3. Illustration de l'usage de noms de variables plus longs pour des expressions de plusieurs mots.

céder ainsi pour afficher ces messages si vous ne voulez vous en servir qu'une fois, mais quand on utilise continuellement une expression dans un programme, c'est une méthode de programmation qui évite de passer son temps à la retaper!

Il y a une autre chose à noter ici : à la ligne 40, les deux identificateurs de variables sont séparés par un point-virgule. Même cette séparation n'est pas indispensable au fonctionnement du programme, mais elle est une aide précieuse quand on le relit.

Quand vous utilisez le point-virgule de cette manière, comme séparateur de variables, ou si vous tapez simplement les variables les unes après les autres, l'affichage à l'écran tentera d'éviter de couper les expressions en bout de ligne. Si l'expression "SUPER$" était affichée juste après "TRUC$" sur la même ligne, le mot "informatique" serait coupé, comme le montre la ligne 40. Évidemment, si vous assignez à une variable un texte qui occupe plus d'une ligne d'écran, cette méthode de jonction d'expressions n'empêchera pas la coupure de mots. A la ligne 50, les expressions ont été fondues en une seule grâce au signe + qui les relie. Nous reviendrons sur cet usage de + ultérieurement

Chaînes et nombres

Une variable comme A$ ne se confond pas avec une variable numérique comme A, parce que le nom de la variable chaîne est distingué par le signe $. On peut en fait utiliser les deux variables dans le même programme, puisque l'ordinateur ne se trompera pas entre les deux, lui!

La *figure 3.4* illustre le fait que la différence est plus profonde

qu'il n'y paraît à première vue. Les lignes 10 et 20 assignent des variables numériques, A et B, et des variables chaînes A$ et B$; vous ne pouvez distinguer A de A$, ni B de B$. La seule différence notable est que, lorsque vous les voyez affichées en colonnes, les variables numériques ont toujours un espace devant leur valeur.

```
10 A=2:B=3
20 A$="2":B$="3"
30 CLS
40 PRINT A,B
50 PRINT A$,B$
60 PRINT A" fois"B" font";A*B
70 PRINT A$" fois "B$" est impossible!"
```

Fig. 3.4. Des variables chaîne et nombre peuvent avoir la même apparence une fois affichées, mais elles sont différentes !

La différence apparaît lorsque l'ordinateur essaye de faire un calcul arithmétique. Il peut multiplier deux variables numériques parce que des nombres peuvent être multipliés, mais il ne peut multiplier deux variables chaînes, qu'elles représentent des nombres ou pas. Vous pouvez multiplier "2" par "3", mais pas "2 RUE DES CHATAIGNIERS" par "3 RUE DES ACACIAS". L'ordinateur refuse donc d'effectuer multiplication, division, addition, soustraction ou toute autre opération arithmétique sur des chaînes. La tentative d'effectuer une opération interdite à la ligne 70 entraîne un message d'erreur à l'exécution, et ce type d'erreur interrompra toujours un programme. Le message qui apparaît est "Type mismatch in 70" (Mélange de types à la ligne 70), qui signifie que vous avez essayé de faire avec des chaînes une opération qui ne peut être effectuée qu'avec des nombres ou des variables numériques.

Nous verrons plus tard des opérations qu'on ne peut effectuer que sur des chaînes, et non sur des nombres ; par conséquent, tenter d'effectuer ces opérations sur des nombres ou des variables numériques entraînera le même message d'erreur. La différence entre les deux types est importante. L'ordinateur stocke les nombres d'une manière très différente des chaînes. Cette différence a pour but de faciliter —

pour l'ordinateur, cela s'entend — l'utilisation de l'arithmétique pour les variables numériques, et d'autres opérations sur les chaînes.

```
10 A$="LATOUR"
20 B$="MAUBOURG"
30 CLS
40 PRINT "Dites-moi juste "A$+"-"+B$", d
it-il."
50 PRINT:A$="123":B$="456"
60 PRINT"La chaine liée est ";A$+B$
70 PRINT"L'addition donnerait";579
```

Fig. 3.5. Concaténation ou jonction de chaînes. Ce n'est pas la même opération qu'une addition !

On peut faire une opération sur les chaînes, qui paraît arithmétique, mais elle est impossible sur les nombres. Cette opération utilise le signe +, mais ce n'est pas une addition au sens d'ajouter des nombres. La *figure 3.5* illustre cette action de fusion de chaînes, qu'on appelle souvent *concaténation.* Cela n'a rien à voir avec l'arithmétique, comme vous le verrez grâce aux lignes 40 et 60. La ligne 60 utilise des nombres à la place des noms placés entre guillemets. Cela en fait de véritables noms de nombres, écrits en chiffres. Juste pour mettre en évidence la différence, la ligne 70 montre ce qui se serait passé si on avait eu des variables numériques. La concaténation est un moyen très utile d'obtenir des chaînes qui exigeraient autrement beaucoup de dactylographie. Vous avez d'ailleurs déjà vu comment elle force deux chaînes à se relier.

Jetez un coup d'œil à la *figure 3.6.* Elle définit A$ et B$ comme des caractères qui peuvent être utilisés pour encadrer

```
10 A$="***":B$="###"
20 S$=" LE NOUVEAU CPC464 "
30 CLS
40 PRINT TAB(5)A$+B$+S$+B$+A$
```

Fig. 3.6. Utilisation de la concaténation pour encadrer un titre.

un titre. Le titre est défini à la ligne 20 comme 'LE NOUVEAU CPC464'. La ligne 40 affiche une chaîne concaténée.

A côté de cette histoire de concaténation, il y a une instruction très utile qui vous fera une chaîne au moyen d'un nombre quelconque de caractères identiques. Cette instruction est STRING$, qui doit être suivie de deux items entre parenthèses. Le deuxième item est le caractère que vous voulez utiliser, et le premier est le nombre de caractères que vous désirez. Par exemple, si vous programmez G$ = STRING$(20,"$"), cela aura pour effet que G$ contienne vingt signes dollar.

```
10 CLS
20 A$=STRING$(10,"*")
30 B$=STRING$(5,"#")
40 PRINT:PRINT A$+B$+" TITRE "+B$+A$
```

Fig. 3.7. Utilisation de STRING$ pour fournir un ensemble de caractères identiques. Voir la *figure 8.10* pour un autre usage de STRING$.

La *figure 3.7* illustre comment utiliser la fonction STRING$ pour encadrer un titre.

En prenant des données avec INPUT

Jusqu'ici, tout ce qui a été affiché sur l'écran par un programme devait avoir été placé dans le programme préalablement à son exécution. Nous ne sommes pourtant pas condamnés à une pareille restriction, car l'ordinateur nous permet d'user d'un autre moyen d'introduire des informations dans un programme, pendant son exécution (peu importe que ces informations soient des nombres ou du texte). Ce genre d'introduction d'information est appelé un INPUT (une entrée) et l'instruction BASIC qui lui est consacrée se trouve aussi être INPUT.

La *figure 3.8* illustre ce point avec un programme qui affiche votre nom. Mais je ne connais pas votre nom, donc je ne peux l'inscrire au préalable dans le programme. Quand vous exécutez ce programme, les mots :

```
10 CLS
20 PRINT"Quel est votre nom"
30 INPUT NAME$
40 CLS:PRINT:PRINT
50 PRINT NAME$;" -c'est vous que ça rega
rde!"
```

Fig. 3.8. Utilisation de l'instruction INPUT. Le nom que vous tapez est mis dans l'expression de la ligne 50.

Quel est votre nom

s'inscrivent sur l'écran. Sur la ligne d'en dessous vous verrez un point d'interrogation, suivi du curseur (jaune). L'ordinateur attend alors que vous tapiez quelque chose, et que vous appuyiez sur ENTER. Tant que la touche ENTER ne sera pas enfoncée, le programme restera à la ligne 30, à vous attendre. Si vous êtes honnête, vous taperez votre vrai nom, tapez-le simplement comme vous voulez le voir affiché. Quand vous appuyez sur ENTER, votre nom est assigné à la variable NOM$. C'est une méthode d'assignation à une variable chaîne qui n'exige pas de guillemets. Le programme peut ensuite se poursuivre, si bien que la ligne 40 nettoie l'écran et crée deux lignes d'espace. La ligne 50 affiche ensuite l'expression célèbre, avec votre nom au début.

Vous pouvez évidemment avoir répondu Mickey Mouse ou Lucky Luke ou n'importe quoi d'autre à votre guise. L'ordinateur n'a aucun moyen de savoir que l'une ou l'autre de ces réponses n'est pas votre vrai nom. Même si vous ne tapez rien, et vous contentez d'appuyer sur ENTER, il poursuivra l'exécution, sans aucun nom.

Nous ne sommes pas limités à l'usage de variables chaînes

```
10 CLS:PRINT "Donnez un nombre, s'il vou
s plait"
20 INPUT N
30 CLS:PRINT
40 PRINT "Le double de"N" est";2*N
```

Fig. 3.9. INPUT d'une variable numérique. Ce que vous tapez doit être un nombre.

avec INPUT. La *figure 3.9* illustre un usage d'INPUT avec une variable numérique, n. La même procédure est utilisée. Quand le programme s'arrête avec le point d'interrogation, vous pouvez taper un nombre et appuyer sur ENTER. Le fait d'appuyer sur ENTER assignera votre nombre à n, et permettra au programme de continuer. La ligne 40 prouve ensuite que le programme traite le nombre que vous avez fourni. Quand vous utilisez une variable numérique dans une instruction INPUT, ce que vous tapez avant d'appuyer sur ENTER doit être un nombre. Si vous tentez d'introduire une chaîne de caractères, l'ordinateur refusera de l'accepter, et vous obtiendrez un message d'erreur — "Redo from start" (Recommencez depuis le début). A la différence de la plupart des messages d'erreur, celui-ci *n'*arrête *pas* le programme ; il se contente de vous donner une nouvelle chance de fournir une réponse conforme à ce qui est attendu.

Si votre instruction INPUT utilise une variable chaîne, vous pouvez taper *n'importe quoi,* ce sera accepté quand vous appuyerez sur ENTER. Mais vous aurez un message d'erreur si vous essayez de faire des opérations arithmétiques sur la chaîne résultante.

```
10 CLS
20 INPUT"Tapez votre nom, s'il vous plai
t ";NOM$
30 PRINT
40 PRINT"Ravi de vous rencontrer, ";NOM$
```

Fig. 3.10. Utilisation d'INPUT pour afficher une expression qui demande l'introduction de la réponse.

On peut placer INPUT dans un programme de manière à donner l'impression que l'ordinateur fait attention à ce que vous tapez. La *figure 3.10* en montre un exemple — mais avec un usage d'INPUT légèrement modifié. Cette fois-ci, il y a une *expression après l'instruction INPUT.* L'expression est placée entre guillemets, suivie d'un point-virgule et enfin de la variable NOM$. Cette ligne 20 a le même effet que les deux lignes :

```
15 PRINT "Tapez votre nom, S.V.P.";
20 INPUT NOM$
```

et cette fois le point d'interrogation et le curseur apparaissent sur la *même ligne* que la question. Votre réponse est aussi sur la même ligne — à moins que la longueur du nom ne cause un débordement de lettres sur la ligne suivante. Vous pouvez aussi utiliser la virgule dans cette sorte de ligne, à la place du point-virgule. Essayez l'effet par vous-même. La virgule supprime l'affichage du point d'interrogation, mais le curseur reste présent.

L'utilisation d'INPUT n'est pas limitée à un seul nom ou nombre. On peut aussi utiliser INPUT avec deux variables ou plus, et on peut mêler les types de variables sur la même ligne d'INPUT. La *figure 3.11,* par exemple, montre l'usage de deux variables après une seule instruction INPUT. L'une de

```
10 CLS
20 INPUT "Nom et nombre, s'il vous Plait
 ";NOM$,NBRE
30 PRINT:PRINT
40 PRINT "Le nom est ";NOM$
50 PRINT "Le nombre est ";NBRE
```

Fig. 3.11. Assignation de deux variables en un seul INPUT.

ces variables est une variable chaîne NOM$, l'autre est la variable numérique NBRE. Maintenant, quand l'ordinateur arrive à la ligne 20, il affiche le message et puis attend que vous donniez l'*une et l'autre* de ces réponses, un nom et puis un nombre.

Il n'y a qu'une manière correcte de faire cela. C'est de taper le nom, puis une virgule, puis le nombre et enfin d'appuyer sur ENTER. Si vous appuyez sur ENTER après avoir tapé le nom mais avant d'avoir tapé le nombre vous obtiendrez le message « Redo from start » (Recommencez depuis le début), et vous devrez fournir le nom et le nombre, puis appuyer sur ENTER. Le nom et le nombre seront ensuite affichés de nouveau, aux lignes 40 et 50.

Cette expérience vous montre que vous ne pouvez pas introduire de réponse contenant une virgule quand vous utilisez INPUT. Par exemple, si vous avez une instruction INPUT NOM$, vous ne pouvez répondre MACHIN, FRED. Ce serait

considéré comme deux entrées, et seul MACHIN serait accepté. Tenter d'introduire MACHIN, FRED entraînerait le message « Redo from start ». Ils sont très attachés aux détails, ces ordinateurs, et il n'y a pas deux modèles qui aient exactement le même comportement !

Il y a toutefois une autre instruction qui vous permet d'introduire des items qui contiennent des virgules. C'est LINE INPUT, que vous pouvez utiliser exactement dans les mêmes conditions qu'INPUT.

```
10 CLS
20 INPUT "Quatre nombres, s'il vous plai
t ";A,B,C,D
30 PRINT
40 PRINT"Leur somme est ";A+B+C+D
```

Fig. 3.12. Un INPUT qui demande quatre nombres.

On peut étendre ce principe plus loin. La *figure 3.12* demande l'introduction de quatre nombres. Encore une fois, ces réponses doivent être fournies en une seule fois, séparées par des virgules, et vous ne devez appuyer sur ENTER qu'après que les quatre réponses ont été tapées. Les nombres sont assignés aux variables numériques, et le programme affichera la somme avec la ligne 40.

La lecture des DATA

Il y a encore un autre moyen d'incorporer des données dans un programme pendant son exécution. Celui-ci implique la lecture d'items dans une liste, et il utilise deux instructions READ et DATA. Le mot READ fait que le programme prend un item de la liste. La liste de données est repérée en commençant chaque ligne par le mot DATA. Les items de la liste doivent être séparés par des virgules. Chaque fois qu'un item est lu dans une telle liste, un « pointeur » est modifié de manière à ce qu'à la prochaine demande d'un item, ce soit l'item suivant dans la liste, et non le dernier lu, qui soit fourni.

Nous verrons cela plus en détail au chapitre 5, mais pour

l'instant nous pouvons nous initier aux instructions READ et DATA.

```
10 CLS
20 READ J
30 PRINT "ITEM ";J;":";J;
40 PRINT "EST UN ";
50 READ NOM$
60 PRINT NOM$
70 DATA 5,LECTEUR DE DISQUETTES
```

Fig. 3.13. Utilisation de READ et de DATA pour mettre des informations dans un programme.

La *figure 3.13* utilise les instructions d'une manière très simple. La ligne 20 lit un item numérique, le premier de la liste, et l'assigne à la variable J. Cette variable est affichée à la ligne 30, avec un point-virgule pour garder l'affichage sur la même ligne de sorte que l'expression de la ligne 40 la suive. Le point-virgule de la ligne 40 laisse encore l'affichage sur la même ligne, et la ligne 50 lit le nom qui forme le second item de la liste. Celui-ci est assigné à la variable NOM$, et affiché à la ligne 60.

La ligne 70 contient les DATA, un nombre et une expression. Le nombre peut être dactylographié tel quel, et vous pouvez voir aussi que l'expression *n'*a *pas* besoin de guillemets. Rien de ce que vous assignez à une variable chaîne de cette manière n'a besoin de guillemets dans les lignes de DATA. Si vous mettez quand même des guillemets, ils seront ignorés, sans qu'apparaisse de message d'erreur.

Vous devez toujours soigneusement assortir vos READ et vos DATA : si vous utilisez une variable numérique dans une ligne READ, comme READ A, ce qui correspond dans la ligne de DATA *doit* être un nombre. Sinon, le programme s'arrêtera avec un message d'erreur. Ceci montrera que la ligne de DATA est erronée, et vous donnera l'occasion de la corriger. Peut-être est-ce le bon moment pour interrompre votre lecture et vous reporter à l'*appendice A* sur l'édition ! Si vous utilisez une variable chaîne, comme dans READ A$, peu importe que votre ligne de DATA contienne un nombre ou une chaîne. Souvenez-vous que si vous lisez un nombre en utilisant

READ A$, le CPC464 ne vous permettra pas de faire d'opérations arithmétiques sur ce nombre.

Notez qu'il *doit* y avoir un espace derrière le mot DATA, et que vos items de données doivent être séparés par des virgules. Vous ne pouvez utiliser de données qui contiennent des virgules.

Les instructions READ...DATA trouvent leur efficacité dans la lecture de longues listes d'items, au moyen d'une instruction READ répétée. Ces items sont en général des items dont vous avez besoin à chaque exécution du programme, plutôt que des items tels que vous pouvez en taper dans des réponses. Nous n'y sommes pas encore, aussi n'irons-nous pas plus loin pour l'instant.

Particularités numériques

Ce que nous avons fait de programmation jusqu'ici devrait vous avoir convaincu que les ordinateurs ne s'occupent pas que de nombres. Pour certaines applications, cependant, leur aptitude à manipuler des nombres est très importante. Si vous voulez utiliser votre ordinateur pour résoudre des problèmes scientifiques ou techniques, par exemple, ses aptitudes à manipuler les nombres seront bien plus importantes que si vous l'avez acheté pour les jeux, pour le traitement de texte ou même pour la comptabilité.

Il est donc temps de jeter un bref coup d'œil aux possibilités numériques du CPC464. C'est seulement un court aperçu, parce que nous n'avons pas la place d'expliquer ce que font toutes les fonctions mathématiques, tout simplement. En général, si vous comprenez ce qu'un terme mathématique comme *sin*, ou *tan* ou *exp* signifie, vous n'aurez pas de problème à utiliser ces fonctions dans vos programmes. Si vous ne savez pas ce que ces termes signifient, vous pouvez tout simplement ignorer les paragraphes de cette section qui en traitent.

L'action numérique la plus simple et la plus fondamentale est le fait de compter. Compter implique les idées d'*incrémentation*, si vous comptez vers les grands nombres, et de *décrémentation*, si vos nombres diminuent. Incrémenter un nombre signifie lui ajouter 1, le décrémenter signifie lui sous-

```
10 CLS
20 X=5:PRINT "X vaut"X
30 PRINT
40 X=X+1
50 PRINT "ça a changé -"
60 PRINT "X vaut maintenant"X
```

Fig. 3.14. L'incrémentation. L'usage du signe égal pour signifier « devient ».

traire 1. Ces actions sont programmées de manière apparemment confuse en BASIC, comme en témoigne la *figure 3.14.* La ligne 20 fixe la valeur de la variable X comme 5. Cette valeur est affichée en ligne 20, mais la ligne 40 « incrémente X ». Cette opération est réalisée par l'instruction d'étrange allure X=X+1, qui signifie que la nouvelle valeur attribuée à X est 1 plus l'ancienne valeur de X. Le reste du programme prouve que cette opération d'incrémentation de la valeur de X s'est bien effectuée.

L'usage du signe = pour signifier "devient" est un point auquel vous devez vous habituer. Quand le même nom de variable est utilisé de part et d'autre du signe d'égalité, c'est cet usage d'assignation que nous utilisons. On pourrait aussi bien avoir une ligne :

X=X—1

qui aurait pour effet de donner à X une nouvelle valeur, de 1 inférieure à l'ancienne. X a cette fois été *décrémenté.* On pourrait aussi utiliser X=2*X pour produire une nouvelle valeur de X égale au double de l'ancienne, ou X=X/3 pour produire une nouvelle valeur de X égale au tiers de l'ancienne valeur. La *figure 3.15* montre une autre assignation de ce genre, dans laquelle une multiplication et une addition sont utilisées en même temps pour changer la valeur de X.

La *figure 3.16* illustre l'usage de quelques fonctions numériques. En ce sens, une fonction numérique est une instruction qui opère sur un nombre pour produire un autre nombre. La ligne 10 prend 2.5 comme valeur pour X. La ligne 20 affiche ensuite la valeur de X élevé au carré, c'est-à-dire X multiplié par X. C'est programmé en tapant X↑2, le caractère que le CPC464 utilise pour cela se trouve sur la même touche que le signe de la livre (£).

```
10 CLS
20 X=5:PRINT "X vaut"X
30 PRINT
40 X=2*X+4
50 PRINT "ça a changé -"
60 PRINT "X vaut maintenant"X
```

Fig. 3.15. Une assignation plus élaborée, utilisant une *expression.*

Pour obtenir la racine carrée du nombre qui a été assigné à X, on utilise le mot SQR. On pourrait utiliser l'équivalent X↑.5 (X puissance un demi) mais SQR(X) est plus facile à taper, et à se rappeler. Pour les autres racines, comme la racine cubique, vous pouvez utiliser des expressions comme X↑ (1/3) et ainsi de suite. LOG(X) produit le logarithme naturel de X. Ce n'est pas forcément le type de logarithme dont vous pouvez avoir besoin, et pour avoir les logarithmes ordinaires (en base 10), vous devez utiliser LOG10(X).

```
10 CLS:X=2.5
20 PRINT "X au carré vaut";X^2
30 PRINT
40 PRINT "Sa racine carrée vaut";SQR(X)
50 PRINT
60 PRINT "Son logarithme naturel vaut";L
OG(X)
70 PRINT "et son logarithme décimal vaut
";LOG10(X)
```

Fig. 3.16. Utilisation de fonctions numériques.

La *figure 3.17* illustre les fonctions numériques variées que vous pouvez utiliser, avec une brève explication de ce que chacune fait. Certaines de ces fonctions ne vous seront utiles que si vous vous intéressez à la programmation pour des motifs scientifiques, techniques ou statistiques. D'autres, toutefois, se révèlent utiles à des endroits inattendus, comme dans des programmes graphiques.

La *figure 3.17* montre aussi l'ordre de priorité des opérations. Cet ordre garantit que quand vous tapez quelque chose

comme 3+4*5 vous obtenez le résultat logique. Dans ce cas c'est 23, parce que la multiplication est *toujours* effectuée d'abord, et que l'addition lui succède.

ABS (X)	Convertit un signe négatif en positif.
ATN (X)	Donne l'angle (en radians) dont la tangente est X.
BIN$ (X,Y)	Convertit le nombre X en binaire, et complète de zéros jusqu'à une longueur Y.
CINT (X)	Convertit X en entier, en arrondissant éventuellement X.
COS (X)	Donne le cosinus de l'angle X (en radians).
CREAL (X)	Convertit X en nombre « réel », distinct d'un entier.
DEG	Met les angles en degrés.
EXP (X)	Donne la valeur de *e* à la puissance X.
FIX (X)	Enlève la partie décimale de X.
FRE (X)	Donne la quantité de mémoire inutilisée ou réservée.
HEX$ (X)	Convertit X en hexadécimal (base 16).
INT (X)	Donne la partie entière de X (arrondie à l'entier inférieur le plus proche).
LOG (X)	Donne le logarithme naturel de X.
LOG10 (10)	Donne le logarithme de base 10 de X (log décimal).
MAX (X, Y, Z...)	Donne le plus grand nombre de la liste.
MIN (X, Y, Z...)	Donne le plus petit nombre de la liste.
PI	Donne la valeur de *pi*, quotient de la circonférence au diamètre d'un cercle.
RAD	Met les angles en radians.
RANDOMIZE (X)	Commence une nouvelle séquence de nombres « aléatoires ».
RND (X)	Donne un nombre aléatoire compris entre 0 et 1.
ROUND (X, Y)	Arrondit X au nombre de places spécifié par Y.
SGN (X)	Donne le signe de X. Le résultat est +1 si X est positif, —1 si X est négatif, 0 si X vaut zéro.
SIN (X)	Donne le sinus de l'angle X (en radians).
SQR (X)	Donne la racine carrée de X.
TAN (X)	Donne la valeur de la tangente de l'angle X (radians).
UNT (X)	X doit être compris entre 0 et 65536. *UNT* le convertit en un nombre compris entre —32768 et +32767.

Ordre de priorité des opérations

Pour ce qui est de l'arithmétique usuelle, l'ordre de priorité est MDAS — multiplication, division, addition et enfin soustraction. L'ordre complet est le suivant :

1. Élévation à une puissance, notée ↑.
2. Multiplication et division.
3. Addition et soustraction.
4. Comparaison, utilisant =, >, <.
5. AND
6. OR
7. NOT

Fig. 3.17. Les fonctions numériques, avec quelques commentaires. (Ne vous inquiétez pas si vous ne voyez pas ce que font certaines, il est probable que vous n'en avez pas besoin !)

Avec quelle précision ?

Les petits ordinateurs ont tous un problème, celui de la précision des nombres qu'ils manipulent. Vous savez sans doute que la fraction 1/3 ne peut être exprimée exactement sous la forme d'un nombre décimal. On ne peut qu'approcher sa vraie valeur, et le degré de précision de cette approche est fonction du nombre de places décimales que nous sommes prêts à afficher : 0.33 est plus près de 1/3 que 0.3, et 0.333 s'en rapproche encore davantage.

L'ordinateur convertit la plupart des nombres qu'il traite sous la forme d'une partie fractionnaire (une mantisse) et d'un multiplicateur. La partie fractionnaire n'est pas décimale, mais une forme spéciale appelée une *fraction binaire* ; la conversion est rarement exacte. Elle est particulièrement

```
10 CLS
20 PRINT 1/3,2/3
30 PRINT 1/11,10/11
40 PRINT 1/3+2/3,1/11+10/11
```

Fig. 3.18. Comment l'ordinateur traite les nombres « illisibles ». Les très grands ou très petits nombres sont convertis sous *forme standard*.

mal adaptée à des nombres comme 1, 10, 100 et 0.1, 0.01, 0.001, toutes les puissances de 10, en fait. Pour éviter des maladresses comme d'afficher 3-2=0.9999999, l'ordinateur arrondira des nombres de ce genre au nombre supérieur ou inférieur avant de les afficher. Tous les ordinateurs ne le font pas avec le même succès — vous pouvez être content d'avoir acheté un CPC464 ! La *figure 3.18* montre comment le CPC464 prend en compte des fractions comme 1/3, 2/3, 1/11 et 10/11. Les nombres que vous voyez sur l'écran ont été arrondis avec neuf décimales, mais le nombre que l'ordinateur *stocke* doit en avoir bien plus. L'arrondissement fait en sorte que l'addition de fractions donne le résultat correct.

Notation scientifique

Le mode d'affichage de la fraction 1/11 vous montre aussi que l'ordinateur a une méthode alternative d'affichage des nombres. Si vous avez jamais travaillé dans des domaines qui utilisent de très grands ou très petits nombres (physique, chimie, ingénierie), vous connaissez déjà cette représentation. Pour ceux qui ne la connaîtraient pas encore, on l'appelle *forme standard* ou *notation scientifique.*

Un nombre en forme standard consiste en une quantité comprise entre 0 et 10 (mais jamais égale ni à 0 ni à 10), et multipliée par une puissance de 10. Prenez par exemple un nombre comme 132000. Si nous déplaçons le point décimal de cinq positions vers la *gauche* (ce qui équivaut à le *diviser* par dix à la puissance cinq), il devient 1.32. Pour obtenir la valeur correcte, ce nombre devrait être multiplié par 10 à la puissance 5, ou, comme on le note E5. Le nombre 132000 peut donc s'écrire 1.32E5. (On lit E “exposant”).

Essayons un petit nombre, comme 0.00036. Cette fois, c'est 3.6 fois dix à la puissance moins quatre, ou 3.6E-4, avec le signe moins parce que l'on doit déplacer le point décimal de quatre positions vers la *droite* pour obtenir ce résultat. Le CPC464, comme la plupart des petits ordinateurs, accepte et affiche des nombres sous cette forme. La conversion est automatique, et le CPC464 affichera les nombres sous cette forme uniquement quand il aura à le faire, c'est-à-dire quand les nombres exigeraient plus de neuf chiffres à l'affichage.

Variables entières

La plupart des ordinateurs permettent de stocker des nombres compris dans un certain intervalle sous une forme plus précise, appelée une *variable entière.* Un entier, pour ce qui est du CPC464, est un nombre entier dont la valeur reste dans les limites —32768 et 32767 (compris).

Un *identificateur de variable entière* consiste en un nom de variable suivi du signe %. Si vous assignez un nombre à une variable entière, seuls les nombres compris dans l'intervalle correct peuvent être utilisés, et tous les décimaux doivent être écartés. Vous obtiendrez le message d'erreur "Overflow" (Dépassement) si vous essayez de faire quelque chose comme :

A% = 32800

par exemple. La *figure 3.19* illustre l'effet du rejet des parties décimales ; en effet, la variable X, dont la valeur est 3.7, est assignée à X% à la ligne 40, et l'affichage de X% à la ligne 50 donne seulement 4. La partie décimale 0.7 a été arrondie à 1. C'est inhabituel, et beaucoup d'autres ordinateurs, si on leur donnait cette ligne, se contenteraient d'omettre 0.7, en donnant pour X% 3.

Utiliser des variables entières présente un double avantage. Le premier est que toute opération arithmétique, sauf la division, donne sur les entiers des résultats exacts, sans nécessiter d'arrondissement. La division fait exception parce que les parties décimales sont ignorées, comme la ligne 70 le montre dans le programme de la *figure 3.19.*

```
10 CLS:X=3.7
20 PRINT "X est un nombre ordinaire égal
 à"X
30 PRINT "X% est un entier."
40 X%=X
50 PRINT "La valeur de X% est"X%
60 Y%=7/5
70 PRINT "7/5 est"Y%" chez les entiers!"
```

Fig. 3.19. Utilisation des entiers. Ils prennent moins de mémoire, et peuvent être plus précis — sauf pour la division.

Le deuxième avantage des entiers est qu'ils prennent moins d'espace en mémoire. De plus, un programme qui utilise des entiers s'exécute bien plus vite qu'un programme qui utilise d'autres types de variables. Le CPC464 vous permet d'effectuer des divisions entières, en utilisant les fonctions \ et MOD (la touche \ est à côté de la touche SHIFT de droite). Le signe \ signifie le résultat entier d'une division entière. Si vous tapez :

PRINT 7\3

par exemple, vous verrez apparaître comme résultat 2. C'est parce que 7/3 a pour résultat 2 et une partie décimale, et la division entière ignore les parties décimales.

MOD est utilisé pour trouver le reste d'une division entière. Si vous tapez :

PRINT 7 MOD 3

le résultat que vous verrez cette fois est 1. C'est le *reste* après la division entière de 7 par 3. La division entière et MOD trouvent leur usage dans des méthodes de programmation plus avancées que celles que nous traitons dans l'espace limité de ce livre.

Utilisation de fonctions définies

Une autre possibilité particulièrement utile pour le travail mathématique est l'usage des *fonctions définies.* C'est une méthode qui permet d'utiliser à maintes reprises un ensemble de fonctions mathématiques sans avoir à l'écrire plus d'une fois dans le programme.

La *figure 3.20* montre un usage efficace d'une fonction définie. C'est un exemple très simple, et jamais vous n'utiliserez

```
5 DEF FN somme(A,B,C)=A+B+C
10 CLS
20 PRINT "Donnez trois nombres S.V.P."
30 INPUT A,B,C
40 PRINT "La somme est ";FNsomme(A,B,C)
```

Fig. 3.20. Utilisation d'une fonction définie très simple.

une fonction définie pour un problème aussi aisé à résoudre, mais la simplicité rend les choses plus faciles à suivre. Les lignes 20 et 30 vous demandent d'introduire trois nombres. La ligne 40 affiche ensuite une quantité appelée FNsomme (A,B,C). Or, cette formule n'a aucun sens à moins d'avoir été définie *plus tôt dans le programme,* et ce dans une ligne qui commence par DEF FN. Quand vous introduisez cette ligne, vous devez faire suivre DEF FN par le nom sur lequel vous vous êtes décidé. Le reste de la ligne montre sur quelles quantités la fonction définie doit travailler (ses paramètres), et ce qu'elle doit en faire. Dans le cas présent, elle va additionner les nombres qui ont été assignés à A, B et C. Cela doit se faire en une seule ligne.

Vous pouvez considérer DEF FN comme une formule que la machine recherchera et utilisera lorsqu'elle trouvera FN dans le programme. Dans cet exemple, DEF FN a été placé en ligne 5, pour qu'on soit sûr que la machine ait reconnu la fonction définie avant qu'on ne l'utilise. Si vous utilisiez DEF FN sur une ligne 100 dans cet exemple, vous obtiendriez le message d'erreur « Syntax error » à l'exécution de la ligne 40.

```
10 DEF FNhypot(a,b)=SQR(a^2+b^2)
20 CLS
30 INPUT "  Deux cotés d'un triangle  re
ctangle,  S.V.P.";x,y
40 PRINT "Le troisième coté vaut ";FNhyp
ot(x,y)
```

Fig. 3.21. Une autre fonction définie, cette fois pour calculer la longueur du côté le plus long d'un triangle rectangle.

La *figure 3.21* montre un autre exemple de cette utile fonction. Là encore, la définition est donnée au début du programme. FNhypot trouvera la racine carrée de a↑2+b↑2, mais les lignes 30 et 40 utilisent des variables x et y, pour les deux côtés d'un triangle rectangle. Le point est que DEF FN dit à l'ordinateur ce qu'il doit faire d'une paire de nombres, et que *le nom de variable qui leur est donné n'a pas d'importance.* Cela facilite beaucoup la programmation dès que vous dépassez le stade du débutant, et que vous commencez à vous muscler un peu !

Définition des types des variables

A un certain stade, vous pourrez vous trouver aux prises avec un programme dans lequel la plupart des variables sont d'un type donné. Vous pouvez vous épargner bien de la dactylographie en utilisant une autre forme de DEF.

Si vous tapez, par exemple, DEFINT A-Z, toutes les variables qui commenceront par l'une quelconque des lettres de A jusqu'à Z se trouveront être des variables entières, sans que vous ayez à utiliser A%, B%, etc., dans votre programme — A, B, C, etc., suffiront. Vous pouvez utiliser DEFSTR pour définir des lettres comme identificateurs de chaînes, et DEFREAL pour signifier des nombres réels. Vous pouvez aussi mêler ces définitions en utilisant par exemple DEFSTR A—I, O—Z:DEFINT J—N pour faire de toutes les variables commençant par les lettres de J à N des entiers, et de toutes les autres des chaînes. Cela peut vous épargner d'avoir à taper bien des signes $ et %!

4

En vous répétant...

Une des activités particulièrement adaptées aux ordinateurs est la répétition d'un ensemble d'instructions, et tout ordinateur est doté d'instructions consacrées à la répétition. Le CPC464 ne fait pas exception à la règle, et possède plus de ces instructions de répétition qu'il n'est habituel d'en trouver sur des ordinateurs de cette gamme de prix et même dans des gammes supérieures. Nous commencerons par l'une des fonctions les plus simples de ces « répéteurs », GOTO.

Branchements et boucles avec GOTO

GOTO signifie en anglais « aller à », et son effet est tout à fait conforme à ce sens : GOTO envoie le programme à un autre numéro de ligne. Normalement un programme est exécuté dans l'ordre croissant des numéros de lignes d'instructions. Pour parler clairement, cela veut dire en commençant par la ligne de numéro le plus bas et en progressant dans l'ordre des lignes jusqu'à la ligne de numéro le plus grand. Utiliser GOTO permet de rompre cet ordre, de sorte qu'une ligne ou un ensemble de lignes seront exécutés dans le « mauvais » ordre, ou indéfiniment réexécutés.

L'instruction GOTO doit être suivie d'un espace et d'un numéro de ligne; vous recevrez un message d'erreur si l'espace est oublié.

La *figure 4.1* donne un exemple d'une répétition très simple, ou « boucle », comme on l'appelle. La ligne 10 contient une simple instruction PRINT. Quand la ligne 10 a été exécutée, le programme passe à la ligne 20, qui lui donne l'ordre de

retourner à la ligne 10. C'est une *boucle sans fin,* qui aura pour effet de remplir l'écran avec les mots :

Amstrad remplit votre écran!

```
10 PRINT "Amstrad remplit votre écran!"
20 GOTO 10
```

Fig. 4.1. Une boucle très simple. Vous pouvez l'arrêter en appuyant deux fois sur ESC.

jusqu'à ce que vous appuyiez sur la touche ESC pour « rompre la boucle ». Toute boucle qui paraît s'exécuter indéfiniment peut être interrompue en appuyant sur la touche ESC. Celle-ci fait ce qu'elle dit, elle permet d'échapper au programme en interrompant son exécution, mais pas complètement. Si vous appuyez sur toute lettre ou sur un chiffre (ou un espace, ou ENTER), le programme reprendra son exécution au point où il s'était arrêté.

Nous verrons plus tard que c'est utile pour chercher les erreurs qui peuvent se trouver dans un programme. Si vous voulez arrêter complètement le programme, pour l'enregistrer ou le modifier, vous devez appuyer à nouveau sur ESC.

Maintenant essayez une boucle qui montre une activité un peu plus visible. La *figure 4.2* montre une boucle dans laquelle un nombre différent est affiché à chaque fois que l'ordinateur passe sur les instructions de la boucle. On appelle cela *chaque passage dans la boucle.* La ligne 10 met la valeur de la variable N à 10. Cette valeur est affichée à la ligne 20, puis la ligne 30 incrémente la valeur de N. La ligne 40 forme la boucle, de sorte que le programme va faire apparaître un compte ascendant très rapide sur l'écran.

```
10 CLS:N=10
20 PRINT N
30 N=N+1
40 GOTO 20
50 REM UTILISER ESC ESC POUR ARRETER
```

Fig. 4.2. Une boucle qui effectue un compte ascendant très rapidement. Vous devrez utiliser deux fois ESC ici aussi.

Là encore, vous devrez utiliser la touche ESC pour l'arrêter, et cela vous donne l'occasion de voir comment le programme se comportera quand vous appuierez sur une autre touche qu'ESC. Comme auparavant, appuyer une deuxième fois sur ESC interrompra définitivement le programme.

Boucles à compteur : FOR ... NEXT

En fait, une boucle incontrôlée comme celle que nous venons de voir n'est pas exactement recommandable, et GOTO est une méthode de création de boucle qu'il est préférable d'éviter ! Il n'y a parfois pas le choix, mais le CPC464 offre deux autres solutions, dont la première est la boucle FOR...NEXT. Ce type de boucle fait donc usage des deux instructions FOR et NEXT. Les instructions qui sont répétées sont placées entre FOR et NEXT. La *figure 4.3* illustre un exemple très simple de boucle FOR...NEXT en action.

```
10 CLS
20 FOR N=1 TO 10
30 PRINT "Amstrad les bat tous!"
40 NEXT
```

Fig. 4.3. Utilisation d'une boucle FOR...NEXT pour un nombre fixe de répétitions.

La ligne qui contient FOR doit aussi inclure une variable numérique qui est utilisée comme compteur, et les nombres qui contrôlent le point de départ du compte et sa fin. Dans l'exemple, N est la variable compteur, et ses limites sont 1 et 10. NEXT se trouve à la ligne 40, et ainsi tout ce qui se trouve entre les lignes 20 et 40 sera répété.

En l'occurrence, ce qui se trouve entre ces lignes est simplement une instruction PRINT, et l'effet du programme sera d'afficher “Amstrad les bat tous!” dix fois. Au premier passage dans la boucle, la valeur de N est fixée à 1, et le message est affiché. Quand l'instruction NEXT est rencontrée, l'ordinateur incrémente la valeur de N — de 1 à 2 dans ce cas. Ensuite, il vérifie si cette valeur dépasse la limite de 10 qui a été fixée. Si la limite n'est pas dépassée, la ligne 30 est répé-

tée, et le processus continuera jusqu'à ce que la valeur de N excède 10 — nous verrons ce point plus tard. L'effet dans cet exemple est de causer 10 répétitions.

```
10 CLS
20 FOR N=1 TO 10
30 PRINT "Le compte est à ";N
40 FOR J=1 TO 1000: NEXT
50 CLS:NEXT
```

Fig. 4.4. Un programme utilisant des boucles imbriquées, une boucle incluse dans l'autre.

Vous n'êtes pas obligé de limiter cette fonction à une seule boucle. La *figure 4.4* montre un exemple de ce qu'on appelle *boucles imbriquées.* On entend par imbriquées qu'une boucle est entièrement contenue dans une autre. Quand des boucles sont imbriquées de cette manière, on peut décrire ces boucles comme respectivement intérieure et extérieure. La boucle extérieure commence à la ligne 20, et utilise la variable N qui va de 1 à 10. La ligne 30 fait partie de la boucle extérieure, et affiche la valeur atteinte par la variable compteur N.

La ligne 40, cependant, est une autre boucle complète. Celle-ci doit utiliser un identificateur de variable différent, et elle doit, de plus, commencer et s'achever avant la fin de la boucle extérieure. Nous avons utilisé la variable J, et nous n'avons rien mis à faire entre la partie FOR et la partie NEXT. Tout ce que fait cette boucle, donc, est perdre du temps, ce qui garantit qu'un temps sensible sépare les différentes actions de la boucle extérieure. Le dernier effet de la boucle principale est d'effacer l'écran à la ligne 50. L'effet global, donc, est de montrer un compte ascendant sur l'écran, assez doucement pour que vous constatiez les changements, et d'effacer l'écran entre chaque affichage. Dans cet exemple, nous avons utilisé NEXT pour indiquer la fin de chaque boucle. Nous pourrions utiliser NEXT J à la ligne 40 et NEXT N à la ligne 50 si nous voulions, mais ce n'est pas indispensable. De plus, cela a pour effet de ralentir un peu l'ordinateur, bien que l'effet ne soit pas important dans ce programme.

Quand vous utilisez effectivement NEXT J et NEXT N, vous devez faire bien attention à mettre les noms de variables cor-

rects après chaque NEXT. Sinon l'ordinateur s'arrêtera avec une erreur « NEXT missing » (NEXT manquant), pour signaler que les NEXT ne sont pas en accord, dans notre cas. Vous auriez aussi ce message si vous aviez oublié un NEXT.

Il est déjà possible à ce stade de voir combien la boucle FOR...NEXT peut être utile, mais il y a encore bien mieux. Tout d'abord, les boucles que nous avons vues comptent en ordre croissant, en incrémentant la variable compteur. Nous n'avons pas toujours besoin de cela, et nous pouvons ajouter l'instruction STEP à la fin de la ligne FOR pour changer la valeur de variation du compteur. Nous pourrions par exemple utiliser une ligne comme :

FOR N = 1 TO 9 STEP 2

qui ferait prendre à N les valeurs successives de 1, 3, 5, 7, 9. Quand on ne tape pas STEP, la boucle prend toujours des incréments de 1.

```
10 CLS
20 FOR N=10 TO 1 STEP -1
30 PRINT N;"secondes, ⁊  compte à rebour
s"
40 FOR J=1 TO 800:NEXT
50 CLS:NEXT
60 PRINT "MISE A FEU!"
```

Fig. 4.5. Un programme de compte à rebours, utilisant STEP.

La *figure 4.5* illustre une boucle extérieure qui a un *pas* (en anglais step) de —1, si bien que le compte se fait en descendant. N commence avec une valeur de 10, et est décrémenté à chaque passage dans la boucle. La ligne 40 forme à nouveau une temporisation pour que le décompte se fasse à une vitesse civilisée. C'est une méthode particulièrement utile pour ralentir le compte à rebours. Si on veut accélérer le rythme, le plus facile est d'utiliser une variable entière comme J% à la place de J. Mais dans ce cas, nous ne pouvons pas utiliser de pas qui contienne des nombres décimaux comme 0.1.

```
10 CLS
20 FOR N=1 TO 5
30 PRINT N
40 NEXT
50 PRINT "N vaut maintenant ";N
60 FOR N=5 TO 1 STEP -1
70 PRINT N
80 NEXT
90 PRINT "N vaut maintenant ";N
```

Fig. 4.6. Valeur de la variable compteur de boucle après la sortie de boucle.

Parfois, quand on utilise des boucles, on constate que l'on a besoin d'utiliser la valeur du compteur, comme N ou J, après l'exécution de la boucle. Toutefois, il est important de savoir quelle sera cette valeur, et la *figure 4.6* s'en charge. Elle contient deux boucles, une croissante et l'autre décroissante. A la fin de chaque boucle la valeur de la variable compteur est affichée. Cela révèle que la valeur de N est 6 à la ligne 50, après l'exécution de la boucle FOR N = 1 TO 5, et 0 à la ligne 90 après l'exécution de la boucle FOR N = 5 TO 1 STEP —1.

Si vous voulez faire usage de la valeur de N, ou de tout autre variable qui vous aura servi de compteur de boucle, vous devez vous souvenir que cette variable sera modifiée *d'un pas de plus* à la fin de la boucle. Vous pouvez évidemment utiliser des valeurs négatives de N dans les boucles. Si vous utilisez des variables entières comme N% ou J% pour des raisons de rapidité, souvenez-vous qu'il y a des limites aux valeurs des variables entières. Vous ne devez pas utiliser d'entier supérieur à 32767 ou inférieur à —32768. Si vous tentez de sortir de cet intervalle, vous obtiendrez un message d'erreur « Overflow » (Dépassement).

Un des traits les plus utiles de la boucle FOR...NEXT, est qu'on peut l'utiliser avec des variables numériques à la place des nombres. La *figure 4.7* illustre simplement ce fait. Les lettres A, B et C sont assignées comme des nombres de la manière usuelle à la ligne 20, mais elles sont ensuite utilisées dans une boucle FOR...NEXT à la ligne 30. Les limites sont fixées par A et B, et le pas est obtenu à partir d'une expression calculable, B/C.

```
10 CLS
20 A=2:B=5:C=10
30 FOR N=A TO B STEP B/C
40 PRINT N
50 NEXT
```

Fig. 4.7. Une instruction de boucle constituée de variables numériques.

La règle est que vous pouvez utiliser dans une boucle comme celle-ci tout ce qui représente un nombre, ou peut être traité de manière à donner un nombre.

Boucles et décisions de branchement

Il est temps de voir les boucles à l'usage et non plus en simple démonstration. Faire le total de nombres est une application simple. Nous voulons un programme qui nous permette d'introduire les nombres, et que l'ordinateur garde un total courant, en ajoutant chaque nombre au total des nombres précédemment fournis.

D'après ce que nous avons fait jusqu'ici, il est facile de voir comment organiser cela si nous voulons utiliser un nombre défini d'entrées, par exemple une série de dix nombres. Le programme de la *figure 4.8* se contente de cela.

```
10 total =0:CLS
20 PRINT TAB(4)"PROGRAMME TOTALISATEUR D
E NOMBRES"
30 PRINT:PRINT "Donnez chaque nombre com
me requis."
40 PRINT "Le programme donnera le total.
"
50 FOR N=1 TO 10
60 PRINT "Nombre";N;"S.V.P. ";
70 INPUT J:T =TOTAL+J
80 NEXT
90 PRINT:PRINT "Le total est ";TOTAL
```

Fig. 4.8. Un programme totalisateur de nombres pour dix nombres.

Le programme commence par mettre une variable numérique « TOTAL » à zéro. C'est la variable numérique qui servira à contenir le total, et il faut qu'elle commence à zéro. En fait, il se trouve que le CPC464 fait automatiquement cette mise à zéro au début de l'exécution d'un programme, mais c'est une bonne habitude que de s'assurer explicitement que tout ce qui doit commencer à une certaine valeur le fait bien.

Incidemment nous ne pouvons utiliser TO pour cette variable, parce que TO est un mot réservé qui fait partie des instructions FOR..NEXT. Vous obtiendrez un message d'erreur "Syntax error" à l'exécution du programme si vous avez utilisé un mot réservé comme nom de variable.

Les lignes 20 à 40 fournissent des instructions et l'action commence à la ligne 50. C'est le début d'une boucle FOR...NEXT qui répétera les actions des lignes 60 et 70 dix fois. La ligne 60 vous rappelle combien de nombres vous avez déjà fournis en affichant à chaque fois la valeur de N, et la ligne 70 vous permet d'introduire par INPUT un nombre qui est ensuite assigné à la variable appelée J. Ce nombre est ensuite additionné au total dans la deuxième moitié de la ligne 70, et la boucle se répète ensuite. A la fin du programme, la variable TOTAL contient la valeur du total — la somme de tous les nombres qui ont été introduits.

Tout cela est bien beau, mais dans combien de cas voudrez-vous additionner exactement dix nombres ? Il serait bien plus commode de pouvoir arrêter l'action en signalant à l'ordinateur qu'on a fini d'une manière quelconque, peut-être en donnant une valeur comme 0 ou 999. Une valeur de ce type est appelée un *terminateur;* c'est quelque chose qui n'est manifestement pas une entrée normale à utiliser, mais un simple signal. Pour un programme totalisant des nombres, un terminateur de 0 est très commode, parce que s'il est additionné au total, il ne fera pas de différence. Nous avons cependant besoin de détecter ce terminateur, et c'est possible grâce à une nouvelle instruction, IF (Si).

Conditions et branchements conditionnels : IF

IF doit être suivi par une condition. Vous pouvez utiliser des conditions comme IF N=20 ou IF NM$+" DERNIER " à

cette fin. Après la condition, vous pouvez utiliser le mot THEN (Alors), qui doit être suivi de ce que vous voulez faire si la condition est satisfaite, le tout sur la même ligne. Vous pouvez simplement vouloir que la boucle s'arrête quand la condition est vraie. Cela peut se programmer en plaçant un numéro de ligne après THEN. Si ce numéro est celui de la dernière ligne du programme, celui-ci s'achèvera quand la condition sera vraie.

Maintenant, si tout cela paraît plutôt compliqué, jetez un coup d'œil à l'illustration simple de la *figure 4.9.* On ne peut utiliser ici une boucle FOR...NEXT parce qu'on ne sait pas combien de fois on voudra passer dans la boucle ; par conséquent nous avons utilisé IF...THEN pour contrôler cette boucle. Les instructions apparaissent en premier, et l'on met la variable TOTAL à zéro en ligne 40. Puis à chaque fois que vous tapez un nombre en réponse à la demande de la ligne 50, il est ajouté au total à la ligne 60, et la ligne 70 affiche la valeur du total à ce stade. La ligne 80 constitue le contrôleur

```
10 CLS:PRINT TAB(10) "Un autre totalisate
ur"
20 PRINT:PRINT "Le programme totalisera
les nombres pour"
30 PRINT "vous jusqu'à ce que vous donni
ez un zéro"
40 TOTAL=0
50 INPUT "Nombre, S.V.P. ";N
60 TOTAL=TOTAL+N
70 PRINT "Le total jusqu'ici est de ";TO
TAL
80 IF N<>0 THEN 50
90 PRINT "FIN DE LA TOTALISATION"
```

Fig. 4.9. Un programme de total courant qui ne peut utiliser FOR...NEXT. La ligne 80 effectue le test qui décide de boucler ou pas.

de boucle. IF est utilisé pour faire le test, et dans ce cas le test consite à trouver si N *n*'est *pas* égal à zéro. Si N n'est pas égal à zéro, on retourne à la ligne 50. Le signe d'apparence étrange < > qui est fait en combinant les signes inférieur à et supérieur à est utilisé pour signifier « non égal ». Vous pouvez mettre un GOTO après THEN, ou l'omettre à votre gré. Comme cela revient à taper plus de texte, je l'ai omis.

L'effet de cette ligne est donc que si le nombre tapé à la ligne 50 n'est pas zéro, la ligne 80 renverra le programme à la ligne 50 à nouveau, pour obtenir un nouveau nombre. Et cela continuera jusqu'à ce que vous introduisiez un zéro. Quand cela se produit, le test de la ligne 80 n'est plus satisfait et le programme passe à la ligne 90. Cette ligne annonce la fin du programme, et comme il n'y a plus de lignes à exécuter, le programme s'arrête.

Si vous appuyez sur ENTER sans avoir tapé de nombre à la ligne 50, le programme considère cela comme équivalent à introduire un zéro, et s'arrête. Toutes les machines n'ont pas un comportement aussi raisonnable !

Cet exemple utilise un seul test avec son IF, mais ce n'est pas une limite. Vous pourriez par exemple décider d'arrêter si on répond 0 ou 999. Dans ce cas votre ligne devrait être :

IF N < > 0 AND N < > 999 THEN 50

ce qui fait deux conditions. Vous pouvez aussi utiliser le mot OR (Ou) quand vous faites un test, tel que :

IF X=0 OR X=999 THEN...

mais vous devriez faire attention, dans un programme de totalisation de nombres, à ne pas additionner 999 à votre total ! Il faut placer certains tests avec précaution !

Signe	Signification
=	Égalité stricte.
>	Quantité de gauche supérieure à celle de droite.
<	Quantité de gauche inférieure à celle de droite.
Ces signes peuvent être combinés comme suit :	
<>	Quantités différentes.
>=	Quantité de gauche supérieure ou égale à celle de droite.
<=	Quantité de gauche inférieure ou égale à celle de droite.
Note : Quand les signes < ou > sont combinés avec =, le signe < ou > doit être utilisé en premier. Utiliser une combinaison comme => sera toujours à l'origine d'un message d'erreur.	

Fig. 4.10. Les signes mathématiques utilisés pour comparer nombres et variables numériques.

La *figure 4.10* illustre le type de test que l'on peut effectuer avec IF. Ces tests utilisent les signes mathématiques par com-

modité, mais vous devez vous souvenir que tous ces signes auront aussi une signification avec les chaînes de caractères. Pour le moment, la signification de = et < > est claire, mais plus tard nous verrons comment utiliser < et >.

Et sinon ? : l'instruction ELSE

IF...THEN forment un test qui peut être très utile dans les programmes. Mais il y a une autre extension d'IF...THEN. Vous pouvez utiliser le mot ELSE (Sinon) pour effectuer un autre test et accomplir une autre action. Un exemple rend cela beaucoup plus clair, aussi regardez la *figure 4.11.*

```
10 PRINT TAB(16)"PILE OU FACE"
20 PRINT "(Taper E pour interrompre)"
30 N=1+INT(2*RND(2))
40 IF N=1 THEN PRINT"FACE" ELSE PRINT "P
ILE"
50 PRINT "Taper E pour interrompre"
60 INPUT A$:IF A$="E" THEN END ELSE 30
```

Fig. 4.11. Un programme simple de pile ou face, où ELSE sert à fournir la deuxième branche de l'alternative.

C'est un simple jeu de pile ou face, sans score. Les lignes 10 et 20 mettent les choses en ordre, comme d'habitude, tandis que la ligne 30 commence la principale boucle de répétition. C'est la ligne importante pour le pari. RND signifie *" tirer au sort "*, et quand l'instruction est suivie d'un nombre positif entre parenthèses, elle signifie que la machine va prendre un nombre décimal au hasard, compris entre 0 et 1. Le nombre n'est jamais 0 ni 1. En multipliant ce nombre par 2, on obtient un nombre qui peut être presque 0, ou presque 2, comme 1.999999. En prenant INT, la partie entière, ou a un nombre entier, soit 0, soit 1. Si on ajoute 1 à ce nombre, on obtient un nombre qui peut être soit 1 soit 2, et c'est ce que nous voulons pour un choix à pile ou face. Le test est fait à la ligne 40, de sorte que si N est égal à 1, on affiche le mot " FACE ", et sinon on affiche " PILE ". Dans cet exemple, ELSE est utilisé pour choisir la deuxième branche de l'alternative. ELSE est utilisé à nouveau à la ligne 60 pour décider s'il faut mettre fin au programme ou non.

Maintenant à vous de transformer ce programme en un jeu plus actif, en demandant à l'utilisateur de deviner ce qui va être tiré au sort, et en notant le score!

L'importance de ELSE est que cette instruction vous offre une option. Si un test est satisfait, quelque chose peut être effectué (l'affichage d'un message par exemple), et avec ELSE une autre tâche (un autre message, par exemple) peut être engagée, si le test n'est pas satisfait (et seulement s'il n'est pas satisfait).

Plus tard nous verrons comment programmer un plus grand nombre de tests. Pour le moment, il est temps de voir un autre type de boucle, qui vous permet bien plus de choix quant à la manière de sortir de la boucle.

Boucles " Tant que " : WHILE...WEND

Très peu d'ordinateurs familiaux vous offrent plus qu'une simple boucle FOR...NEXT; certains n'ont même pas l'instruction ELSE. Le CPC464, lui, vous offre aussi un type très différent de boucle — les boucles WHILE...WEND (Tant que). Les boucles " tant que " rendent souvent beaucoup plus facile la programmation, et évitent l'utilisation de GOTO, et même de IF...THEN. Le principe est de commencer votre boucle par une condition; puis vous avez autant de lignes que vous voulez pour écrire ce qui doit être fait dans la boucle, et finalement le mot WEND (de WHILE et END) marque la fin de la boucle.

```
10 CLS:PRINT TAB(12)"ENCORE DES TOTAUX":
TOTAL=0:N=1
20 PRINT:PRINT"Donnez un nombre à totali
ser,ou 0 pour":PRINT "interrompre"
30 WHILE N<>0
40 INPUT N
50 TOTAL=TOTAL+N
60 PRINT "Le total jusqu'ici est de ";TO
TAL
70 WEND
80 PRINT "Fin du Programme"
```

Fig. 4.12. Un programme totalisateur qui utilise une boucle WHILE...WEND.

Un exemple serait certainement profitable, aussi jetez un coup d'œil à la *figure 4.12.* C'est une nouvelle version d'un vieil ami, le programme totalisateur de nombres. Cette fois, en même temps qu'on met TOTAL à zéro, on a N=1 près du début. C'est nécessaire à cause du fonctionnement de la boucle WHILE...WEND, comme nous le verrons. Le début de la boucle est à la ligne 30, WHILE N < > O. Cette ligne signifie que la boucle se répétera aussi longtemps que N sera différent de zéro. Quand le programme commence, toutefois, vous n'aurez encore fourni aucun nombre à ce stade. C'est pourquoi une valeur " postiche " de N doit être incluse à la ligne 10 avant le début de la boucle. Sans cette précaution d'écrire N=1, le programme s'achèverait dès son passage à la ligne 30 !

Les démarches de la boucle sont familières, et il n'est pas nécessaire de revenir dessus. Le point important à noter se trouve à la ligne 70, WEND. Cela marque la fin de la boucle, et renverra automatiquement le programme au test WHILE. Dans ces conditions, il n'y a pas besoin de IF ni de THEN, et vous pouvez même imbriquer des boucles WHILE...WEND, comme l'exemple plus compliqué du manuel le montre. La seule difficulté est que le test est fait juste au début de la boucle. Vous devez avoir une valeur pour ce qui est testé à ce stade, sinon la boucle ne s'effectuera pas du tout.

```
10 CLS
20 A$=" "
30 WHILE A$<>"X"
40 PRINT A$
50 READ A$
60 WEND
70 DATA Volnay,Chambertin,Clos,Vougeot
80 DATA Pommard,Romanée-Conti,X
```

Fig. 4.13. Un autre exemple de WHILE...WEND, utilisant READ...DATA pour lire une liste de données.

Regardez un autre exemple, la *figure 4.13,* qui utilise cette fois READ...DATA à l'intérieur de la boucle. C'est un programme qui lit simplement un certain nombre d'items de données dans une liste, jusqu'à ce qu'il rencontre un " X ". A la ligne 20, la variable A$ reçoit pour valeur un espace,

obtenu en tapant des guillemets, la barre d'espacement, puis d'autres guillemets. La boucle débute à la ligne 30, avec la condition, et continue jusqu'à ce qu'on trouve un « X » comme valeur de A$. La boucle consiste à afficher la valeur de A$ (un espace au premier passage), puis à lire la liste de DATA pour trouver une autre valeur de A$. La ligne 60 est le WEND de cette boucle, qui renvoie la valeur de A$ au test de la ligne 30. L'effet global est que le programme affiche la liste de DATA. Très savoureux !

```
10 INPUT "Taper un nombre de 1 à 5 ";N
20 WHILE N<1 OR n>5
30 PRINT "Réponse inacceptable -de 1 à 5
 seulement"
40 INPUT N
50 WEND
60 PRINT "Vous avez choisi"N
```

Fig. 4.14. Utilisation d'une boucle WHILE...WEND dans un piège à inepties pour entrée numérique.

La *figure 4.14* montre encore un autre usage de la boucle WHILE...WEND. Dans ce cas, elle se comporte comme un *piège à inepties*. Un piège à inepties (le terme poli est *validateur de données*) est un morceau de programme qui teste les entrées. Si ce que l'utilisateur a fourni est inacceptable, comme un nombre hors de l'intervalle défini, le piège refuse d'accepter l'entrée, explique par un message pourquoi l'entrée est inacceptable, et donne une autre occasion de répondre. Les pièges à inepties sont très importants dans les programmes où une entrée incorrecte pourrait interrompre le déroulement avec un message d'erreur.

Pour un expert (vous, une fois que vous aurez lu ce livre jusqu'au bout !) ce n'est pas un problème ; un habile GOTO rentrera dans le programme. Mais pour l'utilisateur inexpérimenté, le message d'erreur paraît définitif, et il est probable que tout le travail déjà effectué sera perdu, même si cela a pris toute une journée pour introduire les données !

Dans cet exemple, donc, vous êtes invité à fournir des nombres dans l'intervalle de 1 à 5, limites comprises. Si le nombre que vous fournissez est dans l'intervalle, tout va bien, mais sinon (essayez !) la boucle WHILE...WEND entre en

action. Elle affiche un message d'erreur de votre cru, et vous offre l'occasion de rectifier votre réponse. C'est le propre d'un bon piège à inepties, et les boucles WHILE...WEND sont idéales pour former ce genre de pièges.

Notez, en dépit de l'accent mis sur les nombres dans quelques exemples, que les boucles WHILE...WEND sont aussi à l'aise avec les chaînes de caractères. Vous pouvez avoir des lignes comme :

WHILE NOM$ < > " X "

pour vous permettre d'introduire une liste de noms, ou

WHILE REP$" < > O " AND REP$ < > " N "

pour faire un piège à réponse " O " ou " N " (oui, non).

Réponse à un seul caractère : INKEY$

Jusqu'ici nous avons introduit des réponses en O ou N avec l'usage d'INPUT, qui implique de taper un caractère puis d'appuyer sur ENTER. Cela offre l'avantage de laisser place au remords, parce que l'on peut effacer ce qu'on a tapé et changer une lettre avant d'appuyer sur ENTER. Pour des réponses plus vives, cependant, il y a une autre solution sous la forme de l'instruction INKEY$.

INKEY$ est une instruction qui examine le clavier pour voir si une touche est enfoncée. Cet examen est très rapide, et normalement la seule manière de l'utiliser consiste à placer l'instruction INKEY$ dans une boucle qui se répétera jusqu'à ce qu'une touche soit pressée. La *figure 4.15* montre l'usage d'une telle boucle pour produire un effet « Quand vous serez

```
10 CLS
20 PRINT "Appuyez sur n'importe quelle t
ouche..."
30 WHILE INKEY$="":WEND
40 PRINT "FIN"
```

Fig. 4.15. Utilisation d'INKEY$ dans une boucle WHILE...WEND pour vérifier si une touche a été enfoncée. Certaines touches feront avancer le programme, mais n'afficheront rien sur l'écran.

prêts... ». La boucle WHILE...WEND est très simple, et se répétera tant qu'INKEY$ sera vide. La chaîne vide est produite en tapant une paire de guillemets sans rien entre. Chaque fois que l'ordinateur rencontre INKEY$ il examine le clavier pour voir si une touche est enfoncée. Si aucune ne l'est, INKEY$ est vide, et la boucle WHILE...WEND continue. Quand vous appuyez sur une touche, cependant, la boucle est interrompue et le programme continue. Cette procédure est utile à placer après un ensemble d'instructions. L'utilisateur dispose alors d'autant de temps que nécessaire pour lire les instructions, et peut appuyer sur n'importe quelle touche pour entamer les choses sérieuses. Comme d'habitude, « n'importe quelle touche » signifie en fait tout caractère, car plusieurs touches, comme SHIFT et CTRL, n'ont pas d'effet, et ESC interrompra le programme.

```
10 PRINT "S.V.P. Taper O ou N"
20 K$=INKEY$: IF K$="" THEN 20
30 IF K$<>"N" AND K$<>"Y" THEN PRINT " R
éponse incorrecte - O ou N seulement, S.
V.P."
40 PRINT "Votre réponse est ";K$
```

Fig. 4.16. Utilisation d'INKEY$ pour saisir une réponse « O » ou « N ».

L'instruction INKEY$ produira une chaîne quand une touche est enfoncée, aussi pouvons-nous assigner INKEY$ à une variable chaîne, K$. De cette manière, quand une touche est enfoncée, la quantité qu'elle représente sera assignée à K$, et nous pourrons tester cette chaîne à notre gré. La *figure 4.16* montre l'usage d'INKEY$ de cette manière, pour obtenir une réponse " O " ou " N ", avec un piège à inepties. A la ligne 20, INKEY$ est mis dans une boucle, et est continuellement testé. C'est seulement quand une touche est enfoncée que K$ reçoit une valeur autre que la chaîne vide, et alors la boucle est interrompue. La valeur de K$ est ensuite à nouveau testée, pour voir si la réponse est acceptable.

Un morceau de programme comme celui-ci n'entre pas si facilement dans une boucle WHILE...WEND, et serait très maladroit sous cette forme. La version plus simple, avec IF...THEN, proposée ici, est bien meilleure.

Il y a une autre méthode, assez différente, de tester l'appui sur une touche du clavier. Cette méthode utilise l'instruction INKEY, et elle est fondée sur l'idée que chaque touche peut fournir un code numérique. Ces codes numériques sont donnés dans le manuel, à l'*Appendice III.* L'intérêt d'INKEY est que ce n'est pas une instruction du type « Appuyez sur n'importe quelle touche » — elle sert à détecter simplement une touche donnée.

```
10 PRINT " APPUYEZ SUR LA BARRE D'ESPACE
MENT POUR CONTINUER"
20 WHILE INKEY(47)<>0:WEND
30 PRINT "ON CONTINUE ICI..."
```

Fig. 4.17. L'instruction INKEY, qui peut détecter l'appui sur une touche spécifique.

La *figure 4.17* montre cette procédure, où INKEY(47) est utilisé pour détecter la barre d'espacement. Vous pouvez utiliser INKEY(47)=0 pour tester l'appui sur la barre d'espacement, ou INKEY(47)=-1 pour tester l'*absence* d'appui sur cette barre. Vous pouvez aussi user de nombres qui vous permettront de détecter quand les touches SHIFT ou CTRL (ou les deux à la fois) sont appuyées en même temps que la touche examinée. Par exemple, INKEY(47)=32 testera l'appui de la barre d'espacement avec celui de SHIFT, et INKEY(47)=160 testera l'appui simultané sur les trois touches.

INKEY est un moyen très utile pour tester une touche, particulièrement dans les jeux, parce que *n'importe quelle touche* peut être détectée, y compris les touches de déplacement du curseur qui représentent des flèches, la touche COPY, la touche DEL et les autres. Même la touche ESC peut être testée de cette manière !

5

Autour des chaînes de caractères

Les fonctions pour chaînes de caractères

Au chapitre 3 nous avons assez brièvement examiné les fonctions numériques. Si les nombres vous excitent, c'est bien, mais il se trouve que les fonctions pour chaînes de caractères sont plus intéressantes à bien des titres. En effet les opérations qui attirent vraiment l'œil et fascinent sur un ordinateur sont fréquemment effectuées en utilisant les *fonctions chaînes*. Qu'entend-on par fonction chaîne, alors ? En ce qui nous concerne, une fonction chaîne est une opération qu'on peut effectuer avec des chaînes de caractères. Je sais que cette définition ne vous éclaire guère, aussi passons aux détails.

Une chaîne pour le CPC464, est une collection de caractères représentée par une *variable chaîne*, un nom qui se termine par le signe dollar. On peut mettre pratiquement autant de caractères que l'on veut dans une chaîne du CPC464 — un maximum de 255 caractères par chaîne, soit plus que ce que la plupart des usagers utiliseront jamais. Comme les autres ordinateurs, le CPC464 stocke ses chaînes de manière spéciale, en utilisant ce qu'on appelle le code ASCII. C'est un sigle pour " American Standard Code for Information Interchange " (Code américain standard pour l'échange d'information) ; le code ASCII (prononcez ASKI) est utilisé par la plupart des ordinateurs. Ne le confondez pas avec les nombres codes utilisés avec INKEY, les codes ASCII sont tout à fait différents. La *figure 5.1* montre le tableau des nombres codes

32		33	!	34	"
35	#	36	$	37	%
38	&	39	'	40	(
41	)	42	*	43	+
44	,	45	-	46	.
47	/	48	0	49	1
50	2	51	3	52	4
53	5	54	6	55	7
56	8	57	9	58	:
59	;	60	<	61	=
62	>	63	?	64	@
65	A	66	B	67	C
68	D	69	E	70	F
71	G	72	H	73	I
74	J	75	K	76	L
77	M	78	N	79	O
80	P	81	Q	82	R
83	S	84	T	85	U
86	V	87	W	88	X
89	Y	90	Z	91	[
92	\	93	]	94	^
95		96	`	97	a
98	b	99	c	100	d
101	e	102	f	103	g
104	h	105	i	106	j
107	k	108	l	109	m
110	n	111	o	112	p
113	q	114	r	115	s
116	t	117	u	118	v
119	w	120	x	121	y
122	z	123	{	124	|
125	}	126	~	127	

Fig. 5.1. Les nombres codes ASCII standard.

ASCII, et les caractères qu'ils produisent sur une imprimante standard.

Chaque caractère est représenté par un nombre, dans l'intervalle entre 32 (qui représente l'espace) et 127. Sur l'imprimante, 127 produit un carré noir, mais sur l'écran du CPC464 vous verrez un motif en damier.

Vous pouvez assigner des caractères à une variable chaîne en utilisant le signe égal. Quand vous procédez à une assignation de cette manière, vous devez encadrer la chaîne de guillemets. Vous pouvez aussi faire des assignations en utilisant INPUT, ou READ et DATA, ce qui vous dispense des guillemets.

Les variables numériques sont représentées avec un codage tout différent, qui utilise un nombre constant de codes quelle que soit la valeur de la variable, petite ou grande. Il y a un type de codage pour les entiers, un autre pour les nombres ordinaires (qu'on appelle *réels*).

Comme une chaîne consiste en un ensemble de nombres codes dans la mémoire de l'ordinateur (un code par caractère) on peut faire sur les chaînes des opérations impossibles sur les nombres. On peut par exemple facilement trouver combien de caractères se trouvent dans une chaîne. On peut choisir plusieurs caractères d'une chaîne, ou les changer, ou en insérer d'autres. Les opérations analogues à celles que nous venons d'énumérer sont ce que l'on appelle les « fonctions chaînes ».

La fonction LEN

Une des fonctions chaînes que j'ai mentionnées sert à trouver combien une chaîne contient de caractères. Puisqu'une chaîne peut contenir jusqu'à 255 caractères, une méthode automatique pour les compter est bien utile, elle s'appelle LEN.

```
10 CLS
20 A$="Vous pouvez donner des extensions
au CPC464"
30 PRINT "Il y a";LEN(A$);" Caractères d
ans cette expression."
40 INPUT "Essayez vous-meme ";B$
50 PRINT B$;" Fait";LEN(B$);" caractères
."
```

Fig. 5.2. Présentation de LEN, une fonction chaîne.

LEN doit être suivi du nom de la variable chaîne entre parenthèses, et le résultat de l'usage de LEN est toujours un nombre, que nous pouvons afficher ou assigner à une variable numérique. Il n'est pas nécessaire de mettre un espace entre le N de LEN et la parenthèse ouvrante.

La *figure 5.2* montre un exemple simple de l'usage de LEN. La ligne 20 assigne une variable, et la ligne 30 vous dit combien il y a de caractères dans la variable. Notez que le mot « caractère » n'a pas le même sens que « lettre ». L'espace, le point, la virgule, etc. comptent comme des caractères pour une chaîne, car chacun est représenté par un code ASCII.

Les lignes 40 et 50 montrent comment vous pouvez trouver la longueur d'une chaîne que vous avez introduite. C'est utile si vous voulez vous assurer qu'un nom entré au clavier n'est pas trop long pour ce que l'ordinateur doit en faire. Vous pourriez par exemple avoir un programme qui place les noms sur un formulaire, avec des colonnes de dimension limitée, comme quinze caractères. Si on entre un nom trop long, il se pourrait qu'il déborde sur une autre colonne. Vous avez sans doute remarqué, si vous avez un nom ou une adresse longs que votre courrier dont l'adresse a été composée par ordinateur en tronque une partie. Désormais vous savez pourquoi !

```
10 CLS
20 T$=" Le remarquable Amstrad"
30 PRINT TAB((42-LEN(T$))/2);T$
```

Fig. 5.3. Utilisation de LEN pour centrer les titres.

Tout cela ne bouleverse pas l'univers, mais nous pouvons en faire bon usage, comme l'illustre la *figure 5.3.* Ce programme utilise LEN dans une routine qui affichera une chaîne appelée T$ centrée sur une ligne. C'est une routine extrêmement utile dans vos programmes, car son usage vous économisera beaucoup de calculs fastidieux à la rédaction de vos programmes. Le principe est d'utiliser LEN pour trouver le nombre de caractères dans T$. Ce nombre est ensuite soustrait de 42, et le résultat est divisé par 2. Si le nombre de caractères dans la chaîne est impair, le nombre de TAB contiendra un .5, mais la partie décimale est complètement ignorée par TAB à

l'affichage. Vous pouvez, incidemment, utiliser 41 ou 42. Quel que soit votre choix, vous constaterez que les mots sont correctement centrés — 42 fonctionne mieux avec les expressions qui ont un nombre pair de caractères, et 41 avec celles qui en ont un impair. Oui, vous pourriez programmer cela avec un petit nombre d'IF...THEN...ELSE...! On peut utiliser une routine de ce type pour centrer n'importe quoi sous le nom de T$.

Au chapitre suivant nous examinerons l'idée de *sous-programme,* qui nous permet de taper un ensemble d'instructions (comme celui pour centrer un titre par exemple) une seule fois, et de l'utiliser pour n'importe quelle chaîne.

STR$ et VAL

Vous savez déjà qu'il y a des opérations que vous pouvez effectuer sur des nombres et non sur des chaînes, et inversement. Cela pourrait être mal commode, mais il se trouve qu'on peut convertir des nombres d'une forme à l'autre très facilement. Cela permet d'effectuer des calculs sur un nombre qui se trouvait sous forme de chaîne, et d'utiliser des fonctions chaînes avec un nombre qui se trouvait auparavant stocké sous forme numérique.

```
10 N$="22.5":V=2
20 CLS:PRINT
30 PRINT N$;" fois";V;" font";V*VAL(N$)
40 PRINT
50 V$=STR$(V)
60 PRINT " Il y a";LEN(V$);" caractères
dans";V" !"
70 PRINT
80 PRINT N$;" ajouté à";V$;" donne ";N$+
V$;" ?"
```

Fig. 5.4. Conversion de nombres en chaînes et inversement : utilisation de STR$ et de VAL. Notez l'espace que STR$ introduit.

Jetez un coup d'œil à la *figure 5.4.* Tout d'abord on fait de N$ une chaîne qui contient un nombre, et de V un nombre sous forme numérique. La ligne 30 montre comment on peut faire

des calculs arithmétiques avec N$. En tapant VAL(N$) à la place de N$ seul, la valeur numérique de N$ est utilisée dans le calcul, et le résultat correct est obtenu.

A la ligne 50, le nombre V est transformé en chaîne, V$. Il y a toutefois un avertissement là, comme la ligne 60 le montre : le nombre V était égal à 2, un nombre à un seul chiffre. Quand STR$ a été utilisé pour convertir V en chaîne, le nombre de caractères s'est accru mystérieusement d'un. C'est à cause de l'espace invisible qui est introduit pour le signe + ou —. La routine STR$ inclut toujours l'espace, de sorte que la longueur d'une chaîne obtenue à partir d'un nombre a toujours un caractère de trop sauf s'il y avait un signe — debant le nombre. La ligne 80 vous rappelle ce qui arrive si vous oubliez VAL, et que vous essayez d'additionner deux chaînes !

Tranchez à gauche : LEFT$

Le groupe d'opérations sur les chaînes que nous allons examiner maintenant concerne les opérations d'*extraction*. Le résultat de l'extraction sur une chaîne est une autre chaîne, copie d'un morceau de la chaîne la plus longue. L'extraction de chaîne permet de trouver quelles lettres ou caractères sont présents à différents endroits dans une chaîne.

```
10 CLS
20 A$="Amstrad"
30 B$="usez-vous à programmer sur une ma
chine professionnelle."
40 PRINT LEFT$(A$,2)+B$
```

Fig. 5.5. Utilisation de l'effet d'extraction de LEFT$.

Tout cela peut paraître plutôt terne, aussi jetez un coup d'œil à la *figure 5.5.* La chaîne A$ est assignée à la ligne 20, et une autre chaîne est assignée à la ligne 30. Ce qui s'affiche à la ligne 40 est une expression qui utilise les deux premières lettres d'AMSTRAD. Maintenant comment cela s'est-il produit ? L'instruction LEFT$ signifie « copier une partie de chaîne commençant du côté gauche ». LEFT$ doit être suivi de deux quantités, entre parenthèses et séparées par une virgule. La

première est le nom de la variable chaîne dont on veut un morceau, A$ dans l'exemple. La deuxième est le nombre de caractères que vous voulez extraire (copier en fait) à partir de la gauche. L'effet de LEFT$(A$,2) est donc de copier les deux premières lettres d'AMSTRAD, ce qui donne AM. On ajoute ensuite l'autre chaîne à la ligne 30, ce qui nous donne l'expression affichée à l'écran à la ligne 40.

```
10 CLS:PRINT:PRINT
20 INPUT "Votre Prénom, S.V.P. ";PN$
30 PRINT:INPUT "Votre nom, S.V.P. ";N$
40 PRINT:PRINT
50 PRINT"Vous serez dorénavant connu com
me ";LEFT$(PN$,1)+"."+LEFT$(N$,1)+". ici
."
```

Fig. 5.6. Extraction d'initiales à l'aide de LEFT$.

Pour un usage plus sérieux de cette instruction, regardez la *figure 5.6.* Ce programme a pour effet d'isoler vos initiales, en utilisant LEFT$ avec un peu de concaténation. Les INPUT des lignes 20 et 30 trouvent vos nom et prénom, et les assignent à des variables, N$ et PN$. La ligne 50 affiche ensuite vos initiales en utilisant LEFT$ pour extraire la première lettre de chaque chaîne. Les lettres sont ensuite assemblées avec des points, en utilisant la concaténation, à la ligne 50.

Si vous avez deux joueurs dans un jeu, il est souvent utile de donner les initiales devant le score, plutôt que le nom en toutes lettres, mais les noms complets peuvent être conservés pour d'autres usages dans le programme.

Et à droite : RIGHT$

L'extraction de chaîne n'est pas limitée à la copie de la gauche d'une chaîne. On peut aussi copier des caractères à partir de la droite d'une chaîne. Cette possibilité n'est pas aussi fréquemment utile que la précédente, mais elle rend quand même des services. La *figure 5.7.* illustre l'usage des instructions qui permettent d'éviter d'avoir à retaper un mot. Il y a des usages plus sérieux que celui-là. Vous pouvez, par

```
10 CLS
20 A$="L'Amstrad est fascinant"
30 PRINT:PRINT
40 PRINT "Tout cela est ";RIGHT$(A$,9);"
 pour moi"
```

Fig. 5.7. Utilisation de RIGHT$ pour extraire des lettres de la droite d'une chaîne.

exemple, extraire les huit derniers caractères d'une chaîne de nombres comme 03-246-7767. J'ai dit une *chaîne* de nombre délibérément, parce qu'un numéro comme celui-ci doit être stocké comme une variable chaîne et non comme un nombre. Si vous essayez d'assigner cela à une variable numérique, vous allez obtenir une réponse inepte. Pourquoi ? Parce que quand vous tapez N = 03—246—7767 l'ordinateur suppose que vous voulez soustraire 246 de 3, puis 7767 du résultat. La valeur de N sera donc — 8010, ce qui n'est pas exactement ce que l'on avait dans l'esprit !
Si vous utilisez N$ = " 03-246-7767 " tout va bien.

```
10 CLS
20 INPUT "Votre nom, S.V.P.";A$
30 N=LEN(A$)
40 FOR J=1 TO N
50 PRINT LEFT$(A$,J);TAB(21)RIGHT$(A$,J)
60 NEXT
```

Fig. 5.8. Extraction à gauche et à droite d'une chaîne.

On peut faire toutes sortes de choses intéressantes avec LEFT$ et RIGHT$. Regardez par exemple la *figure 5.8*; elle fait de drôles d'opérations avec les lettres de notre nom. Le programme vous demande de donner votre nom à la ligne 20, et le nom est assigné à A$. A la ligne 30, on utilise LEN de façon que la variable N contienne le nombre total de caractères de votre nom. Cela comprend les espaces et traits d'union — personne n'est censé utiliser d'astérisques ou de dièse ! La ligne 40 commence une boucle qui utilise comme limite le nombre total de caractères. La ligne 50 constitue l'opération répétée. Quand J est à 1, la ligne 50 affiche la première lettre à la gauche de votre nom sur le bord gauche de

l'écran, et la dernière lettre de la droite de votre nom sur la droite de l'écran. Au passage suivant, une nouvelle ligne est choisie, et deux lettres sont affichées, et ainsi de suite, jusqu'à l'affichage du nom entier.

Si vous utilisez LEFT$ ou RIGHT$ avec un nombre supérieur au nombre de lettres de la chaîne dont vous voulez copier un morceau, vous obtenez tout simplement la chaîne entière.

Le domaine du milieu : MID$

Il y a une autre instruction d'extraction de chaîne qui est beaucoup plus puissante que LEFT$ ou RIGHT$. C'est MID$, qui doit être suivie de trois items, entre parenthèses et séparés par des virgules. Le premier est le nom de la chaîne dont on veut extraire une partie, comme vous pouvez vous y attendre désormais. Le deuxième est un nombre qui spécifie à partir de quel caractère vous voulez commencer l'extraction. Ce nombre est le numéro d'ordre du caractère, compté à partir de la gauche de la chaîne, en comptant le premier caractère comme 1. Le troisième paramètre est un autre nombre, celui qui indique combien vous voulez prendre de caractères, de gauche à droite, à partir de la position spécifiée par le premier nombre.

```
10 CLS
20 A$="Amstrad CPC464"
30 L=LEN(A$)
40 FOR N=1 TO L
50 PRINT MID$(A$,N,1);" ";:NEXT
60 PRINT:PRINT
70 FOR N=1 TO L
80 PRINT MID$(A$,N,1)+"+";:NEXT
```

Fig. 5.9. Utilisation de MID$. Permet d'extraire de n'importe quelle partie d'une chaîne et peut, comme LEFT$ et RIGHT$, être contrôlé par des variables.

C'est beaucoup plus simple à comprendre sur un exemple que sur une description, essayez donc le programme de la *figure 5.9.* La ligne 20 assigne A$ à AMSTRAD CPC464 et la ligne 30 prend L, le nombre de caractères de cette expression. La boucle commence à la ligne 40, et affiche des lettres prises dans l'expression. Avec 1 comme valeur de N, la lettre

extraite est A, parce que sa position dans la chaîne est 1, et l'on copie une lettre à partir de cette position. Si nous avions utilisé MID$(A$,1,2), nous aurions eu AM et si nous utilisons MID$(A$,3,2) nous aurions ST.

En fait, on prend une lettre à la fois, et on affiche un espace. Le point-virgule de la ligne 50 fait en sorte que la lettre extraite ensuite soit affichée sur la même ligne. L'effet est que les lettres sont affichées espacées. La deuxième boucle aux lignes 70 et 80 a le même genre d'effet, mais place un + entre les lettres au lieu d'un espace.

```
10 CLS
20 INPUT "Votre nom, S.V.P. ";N$
30 L=LEN(N$):C=INT(L/2)+1
40 FOR N=1 TO C
50 PRINT TAB(21-N)MID$(N$,C-N+1,N*2-1)
60 NEXT
```

Fig. 5.10. Une pyramide de lettres : l'action de MID$ avec une formule.

Un des traits de toutes ces instructions d'extraction de chaîne est que l'on peut utiliser des variables ou des expressions à la place de nombres. La *figure 5.10* montre une opération d'extraction plus complexe, qui fait intervenir des expressions. Tout commence assez innocemment à la ligne 20, avec la demande de votre nom. Ce que vous tapez est assigné à la variable N$, et à la ligne 30 on effectue un tour de passe-passe mathématique. Comment cela fonctionne-t-il ? Supposons que vous tapiez DONALD comme nom. Ce nom a six lettres, aussi, à la ligne 30, L reçoit la valeur de 6, et C la partie entière de L/2, c'est-à-dire 3, plus 1, ce qui fait 4. La ligne 40 commence ensuite une boucle à quatre passages. Au premier passage, on affiche à TAB(20) (parce que N = 1 et 21—N donne 20) le MID$ du nom, en utilisant C—N + 1 (4—1 + 1 = 4), et N*2—1 (1*2—1 = 1). On affiche donc Mid$(N$,4,1), soit A dans notre exemple. Au passage suivant dans la boucle, N vaut 2, C—N + 1 vaut 3 et N*2—1 vaut aussi 3. On affiche donc MID$(N$,3,3), qui donne NAL. La boucle continue ainsi, et le résultat est que vous voyez sur l'écran une pyramide de lettres provenant de votre nom. C'est tout à fait impressionnant si il est long ! Dans le cas contraire, imaginez-en un plus long !

Le code ASCII : ASC et CHR$

Il est temps de regarder d'autres types de fonctions chaînes. Si vous revenez en arrière de quelques pages, vous vous rappellerez que nous avons introduit l'idée du code ASCII. C'est le code numérique qui est utilisé pour représenter chacun des caractères que l'on peut afficher. On peut trouver le code de n'importe quelle lettre en utilisant la fonction ASC, qui est suivie entre parenthèses par une chaîne de caractères entre guillemets (ou une variable chaîne sans guillemets). Le résultat d'ASC est un nombre, le nombre code ASCII de ce caractère. Si vous utilisez ASC(" CPC 464 ") vous aurez le code de C seulement : ASC ne donne que le code du premier caractère, et ignore les suivants.

```
10 A$="Programmation de l'Amstrad CPC464
"
20 CLS:PRINT
30 FOR N=1 TO LEN(A$)
40 PRINT ASC(MID$(A$,N,1));" ";
50 NEXT
```

Fig. 5.11. Utilisation d'ASC pour trouver le code ASCII des lettres.

La *figure 5.11* montre cette fonction en action. La variable chaîne A$ est assignée à la ligne 10, et à la ligne 30 une boucle commence. Elle va examiner toutes les lettres de A$, une par une, en utilisant MID$(A$,N,1), et le code ASCII de chaque lettre est trouvé avec ASC. Regardez attentivement comment les parenthèses sont utilisées ! L'espace entre les guillemets, et les points-virgules de la ligne 40 font en sorte que les codes soient tous affichés avec un espace entre les nombre, et sans prendre de nouvelle ligne pour chaque nombre. Tout simple, en fait.

ASC a une fonction inverse, CHR$. Ce qui suit CHR$, entre parenthèses, doit être un nombre du code, et le résultat est le caractère dont le code ASCII a été donné. L'instruction PRINT CHR$(65), par exemple, fera apparaître la lettre A, puisque 65 est le code ASCII de A. On peut utiliser cela pour coder des messages. De temps en temps, il est utile de pouvoir cacher un message dans un programme, pour éviter qu'il

soit évident à quiconque lit le listing. Utiliser le code ASCII n'est pas un très bon moyen de cacher un message aux yeux d'un bon programmeur, mais pour un débutant, c'est bien suffisant.

```
10 CLS:PRINT
20 PRINT " Quel est le code pour une inf
ormatique efficace ?"
30 PRINT "Appuyez sur n'importe quelle t
ouche pour reveler le secret"
40 WHILE INKEY$="":WEND
50 FOR J%=1 TO 6
60 READ D%:PRINT CHR$(D%);
70 NEXT J%
100 DATA 67,80,67,52,54,52
```

Fig. 5.12. Utilisation des codes ASCII pour porter un message codé ; utilisation de CHR$ pour obtenir le caractère correspondant à un code numérique.

La *figure 5.12* illustre cet usage. La ligne 40 est une boucle WHILE...INKEY$...WEND pour que le programme attende votre bon plaisir. Quand vous appuyez sur une touche, la boucle qui commence à la ligne 50 affiche 6 caractères sur l'écran. Chacun est lu comme un code ASCII sur une liste, grâce à une instruction READ...DATA dans la boucle. PRINT CHR$(D%) à la ligne 60 convertit les codes ASCII en caractères et affiche ces caractères, en utilisant un point-virgule pour empêcher le saut de ligne. Essayez !

Si vous voulez cacher plus profondément les lettres, vous pourriez utiliser des valeurs comme le quart de chaque code, ou 5 fois le code plus 20, ou ce que vous voulez comme formule. Ces codes transformés pourraient être stockés dans une liste, et la conversion en codes ASCII pourrait être faite dans le programme. Cela dissuadera tous les décodeurs sauf les plus opiniâtres ! Cet exemple, incidemment, illustre l'usage de READ et de DATA dans une boucle. Normalement, on n'utilise READ et DATA que pour des informations que l'on veut conserver dans un programme de cet ordre. Un autre trait est l'usage de variables entières comme J% et D%. Cela prend moins d'espace en mémoire, et utiliser J% dans la boucle la fait avancer sensiblement plus vite.

Relire des données : RESTORE

Tant que nous en sommes à READ et DATA, il existe une autre instruction qui leur est liée. C'est RESTORE. Quand vous utilisez RESTORE tout seul, cela signifie que le pointeur de DATA doit être remis à zéro. Par exemple, si vous venez de lire six items d'une liste de DATA qui en contient seulement six, vous ne pouvez réutiliser READ, parce qu'il n'y a plus rien à lire. Si vous utilisez RESTORE juste avant un autre READ, cependant, vous lirez à nouveau le premier item de DATA. Il y a encore une variante à l'usage de RESTORE. RESTORE suivi d'un numéro de ligne signifie que les DATA seront relus à partir du début de *la ligne mentionnée*. Vous devez évidemment vous assurer que le numéro de ligne choisi correspond bien à une ligne de DATA!

```
10 CLS
20 PRINT"QUELLE LISTE VOULEZ-VOUS?"
30 INPUT" 1,2 ou 3 S.V.P.";N
40 IF n<1 OR n>3 THEN PRINT"ERREUR,REESS
AYEZ":GOTO 30
50 IF  n=1 THEN RESTORE 140
60 IF n=2 THEN RESTORE 150
70 IF n=3 THEN RESTORE 160
80 a$=""
90 WHILE q$<>"X"
100 PRINT q$;" ";
110 READ q$
120 WEND
130 END
140 DATA Opel, Mercedes,Porsche,X
150 DATA Renault, Peugeot,Citroen,X
160 DATA Fiat, Alfa Romeo, Lancia,X
```

Fig. 5.13. Comment utiliser RESTORE pour sélectionner différentes lignes de DATA.

Regardez la *figure 5.13.* Elle propose un choix de données à lire en faisant usage de RESTORE avec un numéro de ligne. Quand vous choisissez un nombre, il est utilisé aux lignes 50 et 70 pour effectuer une instruction RESTORE. Il est dommage que RESTORE n'accepte pas d'expression calculable. Si c'était le cas, on pourrait programmer cela plus proprement, par exemple en écrivant RESTORE 1000*N. Chaque ligne de

DATA contient trois items suivis de X, et la boucle qui lit les DATA est contrôlée de manière à s'arrêter quand on lit X.

On aurait pu utiliser une boucle FOR...NEXT pour la lecture des données, puisque chaque ligne contient le même nombre d'items. Mais en utilisant cette méthode, on peut accepter n'importe quel nombre d'items dans chaque liste, et ces nombres n'ont pas à être identiques dans toutes les listes, ce qui est bien plus commode. RESTORE, utilisé ainsi fournit une méthode utile pour choisir parmi un ensemble de listes celle qui sera utilisée à chaque exécution du programme.

La règle de classement des caractères

Nous avons vu précédemment, à la *figure 4.10,* comment comparer des nombres. On peut aussi comparer des chaînes, en utilisant les codes ASCII comme bases de la comparaison. Deux lettres sont identiques si elles ont des codes ASCII identiques, par conséquent le sens du signe = appliqué aux chaînes de caractères n'est pas difficile à imaginer. Si deux chaînes sont identiques, elles doivent contenir les mêmes lettres dans le même ordre. L'usage de > et < n'est pas aussi évident, tant que l'on ne songe pas au code ASCII. Le code ASCII pour A est 65, et celui de B est 66. De ce point de vue A est « inférieur à » B, parce qu'il a un code ASCII inférieur. Si l'on veut classer des lettres en ordre alphabétique, il suffit de les ranger en ordre de codes ASCII croissants. Cette opération peut être poussée plus loin, pour comparer des mots entiers, caractère par caractère.

La *figure 5.14* illustre cet usage de la comparaison avec les symboles = et >. La ligne 20 assigne un mot sans signification — il s'agit des six lettres du début du clavier alphabétique. La ligne 30 vous demande de taper un mot. Les comparaisons se font ensuite aux lignes 40 et 50. Si le mot que vous avez tapé, et qui est assigné à B$, est *identique* à QWERTY, le message de la ligne 40 s'affiche, et le programme s'arrête. Si QWERTY devait être classé après votre mot dans un index, alors la ligne 50 s'exécuterait. Si par exemple vous tapez PERIPHERIQUE, puisque Q vient après P dans l'alphabet, et a un code ASCII plus grand que P, votre mot B$ vaut moins que A$, et la ligne 50 les échange. Cela se fait en assignant une

```
10 CLS
20 A$="QWERTY"
30 PRINT:INPUT"Tapez un mot, en capitale
s ";B$
40 IF A$=B$ THEN PRINT"LE MEME QUE LE MI
EN!":END
50 IF A$>B$ THEN Q$=A$:A$=B$:B$=Q$
60 PRINT"L'ordre correct est ";A$;" Puis
 ";B$
70 END
```

Fig. 5.14. Comparaison de mots du point de vue de l'ordre alphabétique.

nouvelle chaîne, Q$ à A$ (de sorte que Q$ = " QWERTY "), puis en assignant A$ à B$ (ainsi A$ = " PERIPHERIQUE "), et enfin B$ à Q$ (ainsi B$ = " QWERTY "). La ligne 60 va ensuite afficher les mots dans l'ordre A$ puis B$, qui sera l'ordre alphabétique correct. Si le mot que vous avez tapé vient après QWERTY — par exemple TELEVISEUR — A$ n'est pas « supérieur à » B$, et le test de la ligne 50 n'est pas satisfait. On ne fait pas d'échange, et l'ordre A$ puis B$ reste tel quel, puisqu'il est correct. Notez le point important, à savoir que des mots comme QWERTZ ou QWERTX seront convenablement ordonnés ; ce n'est pas seulement la première lettre qui compte.

Mettez-le sur la liste : les tableaux

```
10 CLS
20 FOR N=1 TO 10
30 A(N)=1+INT(RND(1)*100)
40 NEXT
50 PRINT
60 PRINT TAB(16)"LISTE DES NOTES"
70 PRINT:FOR N=1 TO 10
80 PRINT TAB(2)"Item ";N;" a reçu";A(N);
" comme note."
90 NEXT
```

Fig. 5.15. Un tableau de variables numériques indicées. C'est plus simple que le nom ne le suggère !

Les noms de variables que nous avons utilisés jusqu'ici sont utiles, mais il y a une limite à leur utilité. Supposez par exemple que vous ayiez un programme qui vous permet de taper un grand ensemble de nombres. Comment faire pour assigner un nom de variable différent à chaque item ? La *figure 5.15* illustre une solution. Les lignes 10 à 40 engendrent un ensemble imaginaire de notes d'examen. L'idée est simplement de vous éviter la tâche d'entrer ces données ! La variable de la ligne 30 est une nouveauté. On l'appelle une *variable indicée,* et « l'indice » est le nombre qui est représenté par N. Le terme d'indice n'a rien à voir avec la programmation ; c'est un terme qui était utilisé bien avant les ordinateurs. Combien de fois utilisez-vous des listes dont les items sont numérotés 1,2,3 etc. ? Ces nombres 1,2,3 sont une forme d'indices, mis là simplement pour identifier différents items. De la même manière, en utilisant des noms de variables comme A(1), A(2), A(3), etc., on peut identifier différents items qui ont pour nom de variable commun A. Un membre de ce groupe comme A(2) se désigne comme 2-de-A.

L'utilité de cette méthode est qu'elle permet d'utiliser un seul nom de variable pour une longue liste, en sélectionnant les items uniquement par leur numéro d'indice. Puisque le numéro peut être une variable numérique ou une expression, on peut travailler avec n'importe quel item de la liste.

La *figure 5.15* montre la construction de la liste aux lignes 20 à 40, à l'aide d'une boucle FOR...NEXT. Chaque item est obtenu en tirant au sort un nombre entre 1 et 100, qui est ensuite assigné à A(N). On assigne ainsi dix de ces « notes », puis les lignes 60 à 90 affichent la liste. Cela fait une programmation beaucoup plus nette pour chaque nombre.

Tout va bien jusqu'ici, mais nous avons omis un point. Essayez de modifier la boucle, en écrivant FOR N=1 TO 11, et exécutez le programme. Vous obtiendrez un message d'erreur qui dit "Subscript out of range in 30" (Indice hors des limites en ligne 30). L'ordinateur est prêt à utiliser des indices jusqu'à 10, mais pas plus. Il faut le préparer spécialement à en utiliser de plus grands ; la préparation consiste à réserver de la mémoire pour recevoir les données.

Quand vous utilisez l'instruction DIM (qui signifie dimension), de la mémoire est allouée pour recevoir les éléments de la liste, qu'on appelle aussi *tableau.* Une ligne comme DIM

A(11) vous permet, en fait, de ranger jusqu'à *douze* éléments dans un tableau, parce que l'on peut utiliser A(0) si l'on veut ; mais il ne faut pas essayer d'utiliser A(12), ou tout indice supérieur. Vous obtiendrez toujours un message d'erreur dans ce cas.

L'importante instruction DIM consiste donc à nommer chaque variable que vous utiliserez sous forme de tableau, et à faire suivre le nom du *nombre maximum* (entre parenthèses) d'items que vous utiliserez. On appelle cela déclarer ou *dimensionner* un tableau. Vous n'êtes pas obligé d'utiliser tous les indices mais vous ne devez pas dépasser l'indice maximum. Si vous le faites, le programme s'arrêtera, vous devrez changer l'instruction DIM, et recommencer — ce qui serait désagréable si vous étiez en train de taper une liste de 100 noms ! Notez que vous pouvez dimensionner plus d'une variable dans une ligne DIM, comme le montre la *figure 5.16.*

Même si vous n'êtes pas obligé d'utiliser DIM quand vous uti-

```
10 CLS:DIM A(12),N$(12)
20 PRINT TAB(2)"Entrez des noms et des n
otes S.V.P."
30 FOR N=1 TO 12
40 INPUT"Nom";N$(N)
50 INPUT"Note";A(N)
60 NEXT
70 CLS:TOTAL=0
80 PRINT TAB(16)"LISTE DES NOTES":PRINT
90 FOR N=1 TO 12
100 PRINT TAB(2);N$(N);TAB(22);A(N)
110 TOTAL=TOTAL+A(N)
120 NEXT
130 PRINT
140 PRINT"Moyenne :";TOTAL/12
```

Fig. 5.16. Utilisation de chaînes dans un tableau, et de nombres dans un autre. Les tableaux ont été dimensionnés cette fois.

lisez des indices inférieurs ou égaux à 10, c'est une bonne habitude que de le faire. La raison en est que cela évite une perte d'espace mémoire, en améliorant l'efficacité de l'usage de la mémoire.

La *figure 5.16* étend cet usage des tableaux à un autre aspect.

Cette fois on vous invite à taper un nom et une note pour chacun des douze items. Quand la liste est complète, l'écran est nettoyé, et une variable appelée TOTAL est mise à zéro à la ligne 70. La liste est ensuite proprement affichée, et à chaque passage dans la boucle, le total est fait (à la ligne 110) de sorte que la moyenne peut être affichée à la fin. Le point important ici est qu'on peut conserver sous forme de liste autre chose que des nombres. Le nom correct pour une liste est « tableau », et la *figure 5.16* utilise à la fois un tableau de chaînes (les noms) et un tableau de nombres (les notes).

Rangées et colonnes

Vous pouvez vous représenter un tableau comme une liste d'items, les uns après les autres, mais il existe une autre variété de tableaux, qui permet une différente sorte de liste, appelée une *matrice*. Une matrice est une liste de groupes d'items, où tous les items d'un groupe sont en relation. On peut se représenter une matrice comme un ensemble de rangées et de colonnes, où chaque groupe occupe une rangée, et les items d'un groupe sont répartis sur les différentes colonnes.

```
10 CLS:DIM N$(3,2)
20 FOR N=1 TO 3
30 FOR J=1 TO 2
40 READ N$(N,J)
50 NEXT J:NEXT N
60 FOR n=1 TO 3
70 PRINT TAB(5);N$(N,1);TAB(20);N$(N,2)
80 NEXT
100 DATA Cheval,Poulain,Vache,Veau,
Chien, Chiot
```

Fig. 5.17. Confection d'une matrice de rangées et colonnes.

Jetez un coup d'œil à la *figure 5.17* pour voir comment cela fonctionne. On utilise ici une variable N$ qui a deux indices. Le premier nombre est le nombre de rangées, et le second le nombre de colonnes. On a besoin de deux boucles FOR...NEXT pour introduire les données dans cette matrice. C'est effectué aux lignes 20 à 50. Les items sont ensuite affi-

chés en colonne par la boucle des lignes 60 à 80. Dans cette boucle, la variable N est utilisée comme l'indice de rangée, et l'on utilise les numéros de colonne 1 et 2. Les rangées contiennent des noms d'animaux, et les colonnes séparent les noms que l'on utilise respectivement pour l'adulte et pour le jeune.

Cet exemple utilise une matrice de chaînes, mais une matrice de nombres est aussi possible. Si vous avez fait des mathématiques jusqu'au baccalauréat ou au-delà, vous connaissez sans doute un certain nombre d'exemples d'utilisation de matrices de nombres, en particulier pour la résolution d'équations simultanées.

```
10 CLS:DIM A$(50,2)
20 FOR N=1 TO 10
30 LOCATE 2,5:INPUT "Nom";A$(N,1)
40 LOCATE 2,7:INPUT "Tel. No";A$(N,2)
50 CLS:NEXT
60 CLS:LOCATE 14,1:PRINT "LISTE COMPLETE
"
70 LOCATE 10,4:INPUT"Choisissez une init
iale";J$
80 FOR N=1 TO 10
90 IF J$=LEFT$(A$(N,1),1) THEN PRINT "No
m ";A$(N,1);TAB(20)"Numero : ";A$(N,2)
100 NEXT
```

Fig. 5.18. Utilisation d'un nom et d'un nombre dans une matrice pour une application de répertoire téléphonique.

La *figure 5.18* montre un programme à matrice plus ambitieux. L'idée est de stocker des ensembles de noms et de numéros de téléphone que vous nourrissez au cours de la boucle des lignes 20 à 50. Une fois que la matrice a été remplie, vous pouvez prendre la lettre initiale d'un nom, et demander à l'ordinateur d'afficher le nom et le numéro qu'il a repérés. J'ai omis les pièges à inepties pour laisser l'exemple raisonnablement bref, mais en usage réel vous devriez avoir un piège quelconque, au moins sous la forme d'un message comme :

PRINT « DESOLE, PAS D'ENTREE » ;J$.

Normalement, si vous mettiez au point un programme analogue, vous voudriez aussi l'utiliser plus d'une fois. Il faut une boucle pour pouvoir revenir en 70 chercher un autre nom, après en avoir trouvé déjà un. Vous pourriez aussi vouloir traiter correctement le cas où un nom n'est pas trouvé. Cela se produira lorsque le programme aura accompli la ligne 100 sans être jamais passé sur la partie conditionnée de la ligne 90. On vous donnera des indications à cet effet plus tard !

Recherche d'une sous-chaîne : INSTR

Une autre fonction chaîne utile — et rare — est INSTR. On l'utilise pour trouver si une chaîne est contenue dans une autre. L'instruction s'utilise sous la forme :

X = INSTR(A$,B$)

pour trouver si B$ est contenu dans A$. Si c'est le cas, X reçoit le nombre qui représente la position de la première lettre de B$ qui se trouve dans A$. Si B$ *n*'est *pas* dans A$, X vaut 0. X sera toujours égal à 0 si B$ est plus long que A$. Vous pouvez évidemment toujours utiliser la forme :

PRINT INSTR(A$,B$)

si vous voulez juste voir la position.

La *figure 5.19* montre un exemple simple de l'action de cette fonction. Les lignes 20 à 40 allouent des noms aux chaînes, et

```
10 CLS
20 A$="Albert le Grand"
30 B$="Richard, Bertrand"
40 C$="Ian Sinclair"
50 PRINT "Dans A$, bert est situé à";INS
TR(A$,"bert")
60 PRINT "Dans B$, Bert est situé à";INS
TR(B$,"Bert")
70 PRINT "Dans C$, BERT est situé à";INS
TR(C$,"BERT")
```

Fig. 5.19. Usage de l'instruction INSTR.

les lignes 50 à 70 font les tests, de façon que vous puissiez voir comment ils procèdent. Remarquez que les chaînes doivent être exactes pour que la fonction opère ; ce n'est pas la peine de chercher " Bert " si ce qui est contenu est "bert " ou " BERT ", par exemple.

Pour vous laisser avec des perspectives, supposez que vous ayiez une chaîne A$ = " OUIouiOUAISouaisOKok » et que vous demandiez une réponse oui ou non. Vous pourriez utiliser INSTR pour examiner cette réponse. Si le résultat de X = INSTR(A$,REP$) est zéro, la réponse était tout sauf oui !

6

Menus, sous-programmes et programmes

La *figure 4.16* a introduit l'idée de faire un choix en appuyant simplement sur une touche. Dans cet exemple, le choix de touches était limité à O ou N. Un choix entre deux items est plutôt restreint, mais on peut étendre le choix par une routine de programme appelée un *menu*. Un menu est une liste de choix, d'ordinaire des choix d'options de programme. En prenant l'un de ces choix, on entraîne l'exécution d'une section de programme. Une manière de faire le choix consiste à numéroter les items du menu, et à faire taper le numéro de celui qui doit être exécuté.

La *figure 6.1* montre à quoi ressemble un menu de ce genre sur l'écran. Avec un ordinateur doté d'un BASIC de l'âge de pierre, on utiliserait un ensemble de lignes comme :

```
IF K=1 THEN 1000
IF K=2 THEN 2000
etc.
```

MENU

1. Faire une nouvelle liste de noms.
2. Ajouter des noms à une liste.
3. Lire les noms d'une liste.
4. Supprimer des noms d'une liste.
5. Choisir un nom.
6. Mettre fin au programme.

CHOISISSEZ UN NUMÉRO S.V.P.

Fig. 6.1. Un menu typique, tel qu'il apparaîtrait sur l'écran.

Il y a cependant une méthode bien plus simple, qui utilise les instructions du CPC464 ON...GOTO ou ON...GOSUB. Ces instructions représentent deux manières d'effectuer la même tâche, et nous allons les examiner soigneusement toutes deux, parce que ce genre d'instruction n'existe pas dans quelques versions du BASIC que vous avez pu utiliser.

Pour commencer, supposons que nous voulions choisir un item parmi quatre en tapant un chiffre de 1 à 4 sur le clavier. La première chose à faire est de vous assurer que seuls les nombres de 1 à 4 sont acceptés, et non des nombres comme 0, 5, —10, etc., — en un mot, il faut un piège à inepties. Le deuxième point est que l'usage d'INPUT est un peu fastidieux, parce que vous devez appuyer sur ENTER après avoir tapé la touche du chiffre.

Nous utiliserons donc INKEY$, pour récupérer la réponse (la touche numérique), et il ne restera qu'à vérifier que vous avez choisi un nombre dans l'intervalle défini. Après cette vérification, nous utiliserons le nombre entré pour trouver un numéro de ligne et exécuter le morceau de programme qui y commence.

Dans l'exemple de la *figure 6.2*, les lignes 10 à 60 affichent un titre et un menu, en utilisant des TAB pour faire un affichage à peu près propre. Ensuite vient un conseil sur la manière de choisir (c'est peut-être évident pour vous, mais pas pour tous les utilisateurs) ! Les opérations commencent aux lignes 70 et 80. A la ligne 70, il y a une boucle sur INKEY$, qui durera tant qu'aucune touche n'est enfoncée. Si une touche *est* enfoncée, K$ aura une valeur (un caractère), et la partie ELSE convertit cette chaîne en nombre. La ligne 80 teste ce nombre, pour vérifier qu'il correspond bien à l'intervalle de choix du menu, de 1 à 4.

Si le nombre n'est pas dans l'intervalle, vous obtenez un message poli et un rappel de l'intervalle correct. Si on a appuyé sur une lettre, incidemment, l'expression K=VAL (K$) convertira la lettre en zéro, qui sera rejeté par la ligne 80. N'utilisez jamais un choix numérique 0 dans un menu, car il est beaucoup plus difficile dans ce cas de distinguer lettres et chiffres !

A la ligne 90, avec la valeur de K dans l'intervalle correct, le choix est réalisé par l'expression ON K GOTO 110,130,150,170.

```
10 CLS:PRINT TAB(19)"MENU"
20 PRINT TAB(2)"1. Recette quotidienne"
30 PRINT TAB(2)"2. Situation du stock"
40 PRINT TAB(2)"3. Achats"
50 PRINT TAB(2)"4. Relevé"
60 PRINT:PRINT TAB(4)"Choisissez à l'aid
e d'un nombre de 1 à 4"
70 K$=INKEY$:IF K$="" THEN 70 ELSE K=VAL
(K$)
80 IF K>4 OR K<1 THEN PRINT"Nombre incor
rect -1 à 4- Reessayez.":GOTO 70
90 ON K GOTO 110,130,150,170
100 END
110 PRINT "Ceci est la routine n.1"
120 GOTO 100
130 PRINT "Ceci est la routine n.2"
140 GOTO 100
150 PRINT "Ceci est la routine n.3"
160 GOTO 100
170 PRINT "Ceci est la routine n.4"
180 GOTO 100
```

Fig. 6.2. Un choix par menu utilisant l'instruction ON K GOTO.

Cette liste *doit* être dans l'ordre qui servira au choix du menu. En d'autres termes, la routine qui commence à la ligne 110 devrait répondre aux besoins de l'item 1 du menu, la ligne 130 aux besoins de l'item 2, etc. Évidemment, ces lignes n'ont pas à être numérotées 110, 130, 150, 170. Les numéros auraient aussi bien pu être 7000,600,150,2010, *pourvu que ces numéros correspondent bien au début des différentes routines.*

Dans cet exemple, chaque « routine » consiste en un simple message, suivi de GOTO 60. Cela assure que le programme s'achève après chaque routine. Si nous avions utilisé GOTO 10, nous aurions renvoyé le programme au menu. C'est très souvent ce dont on a besoin. Quand le programme revient toujours au menu, l'un des choix doit être « Fin du programme », de manière à offrir une sortie du programme autre que la pure et simple extinction de l'ordinateur.

La programmation par segments

Beaucoup de programmes consistent en un titre, des instructions, et puis un menu. Suivant le choix que vous faites au menu, une partie du programme est effectuée, et le programme s'achève ou revient au menu. La sélection dans un menu avec une instruction ON...GOTO est utile, mais il y a une meilleure méthode, qui fait usage de *sous-programmes*. Un sous-programme est une portion de programme qui peut être insérée là où vous le voulez dans un programme plus long. Un sous-programme est inséré en tapant l'instruction GOSUB suivie du numéro de ligne où commence le sous-programme. Quand votre programme en vient à cette instruction, il saute au numéro de ligne qui suit GOSUB, tout comme dans un GOTO. Mais, à la différence de GOTO, GOSUB offre un retour *automatique*. Le mot RETURN est utilisé à la fin des lignes de sous-programme, et a pour effet de renvoyer le programme à la ligne (ou l'instruction) qui suit immédiatement GOSUB.

```
10 CLS
20 PRINT"Ceci est une routine ";
30 GOSUB 1000
40 PRINT".":PRINT:PRINT
50 PRINT"  De la lumière rouge et de la
lumière  verte font de la lumière ";:GOS
UB 1000:PRINT"."
60 PRINT:PRINT:END
1000 PRINT "Jaune";
1010 RETURN
```

Fig. 6.3. Utilisation d'un sous-programme — la clé d'une programmation plus avancée.

La *figure 6.3* illustre ce fonctionnement. Quand le programme est exécuté, la ligne 20 affiche une expression, le point-virgule empêche le passage à la ligne. Le GOSUB 1000 de la ligne 30 entraîne l'affichage du mot « JAUNE », et le RETURN de la ligne 1010 renvoie le programme à la ligne 40, l'instruction qui suit immédiatement le GOSUB 1000. Le même effet est produit quand le GOSUB est inclus dans une ligne à plusieurs instructions, comme la ligne 50 le montre. Le GOSUB 1000 aura pour effet d'afficher « JAUNE », et le

RETURN renverra à l'instruction PRINT qui suit GOSUB 1000 sur la ligne 50 ; on ne saute pas à la ligne 60.

```
10 CLS:PRINT
20 PRINT TAB(11)"Choisissez votre monstr
e."
30 PRINT
40 PRINT TAB(2)"1. Vampire."
50 PRINT TAB(2)"2. Loup-garou."
60 PRINT TAB(2)"3. Zombie."
70 PRINT TAB(2)"4. Adjudant."
80 PRINT TAB(2)"5. Assortiment."
90 PRINT:PRINT"Choisissez  un nombre, S.
V.P.":PRINT:PRINT
100 GOSUB 10000:' ROUTINE inkey$
110 IF K<1 OR k>5 THEN PRINT"Erreur de c
hoix -1 à 5-":PRINT "Réessayez.":GOTO 10
0
120 ON K GOSUB 1000,2000,3000,4000,5000
130 PRINT:PRINT"Un autre choix ? tapez O
 ou N."
140 GOSUB 10000:IF K$="O" THEN 10
150 END
1000 PRINT"Du sang, du sang, bon sang":R
ETURN
2000 PRINT"Hurlement,  grondement,  clag
uement de  machoires": RETURN
3000 PRINT"J'obéis,maitre,j'obéis":RETUR
N
4000 PRINT"Trois jours d'arrets de rigue
ur":RETURN
5000 PRINT"Du sang, hurlement, j'obéis,
aux arrets.":RETURN
10000 k$=INKEY$:IF k$="" THEN 10000 ELSE
 k=VAL(k$)
10010 RETURN
```

Fig. 6.4. Un choix par menu faisant usage des sous-programmes.

La *figure 6.4* montre des sous-programmes utilisés comme partie d'un programme de jeu (imaginaire). Les lignes 10 à 80 offrent un choix, et la ligne 90 vous invite à choisir. Les opérations usuelles avec INKEY$ et son piège à inepties suivent. Ensuite la ligne 120 permet d'effectuer le choix. Mais cette fois, le programme reviendra à ce qui suit le choix. Par exem-

ple, si vous avez appuyé sur 1, le sous-programme qui commence à la ligne 10000 s'exécute, et le programme revient à la ligne 130 pour vérifier si vous voulez aussi les sous-programmes qui commencent en 2000, 3000, 4000 ou 5000.

Un sous-programme est très utile pour les choix dans un menu, mais ce procédé est encore plus utile pour des éléments de programme qui doivent être utilisés à plusieurs reprises dans un programme.

A la *figure 6.4,* par exemple, la routine INKEY$ a été écrite comme un sous-programme parce que c'est une routine que vous utiliserez probablement à maintes reprises, dans n'importe quel programme. Intégrer INKEY$ dans un sous-programme signifie que vous n'avez à taper ces lignes de programme qu'une seule fois. Chaque fois que vous avez besoin de ces opérations, vous tapez simplement GOSUB 10000 (ou le numéro de ligne que vous avez utilisé), et la routine sera insérée à l'exécution du programme.

Remarquez que dans chaque cas le sous-programme est placé sur des lignes qui ne peuvent être normalement exécutées : si vous enleviez les instructions GOSUB de ces programmes, les sous-programmes seraient ignorés, parce qu'il y a normalement une instruction END (ou STOP) avant d'arriver au sous-programme. C'est important, car si l'ordinateur arrive par accident sur un sous-programme (sans passer par un GOSUB), le programme s'arrêtera avec un message d'erreur " Unexpected RETURN in 1010 " — ou tout autre numéro de ligne où se trouve le RETURN (RETURN inattendu en 1010). Il faut éviter ce genre d'intrusion en plaçant correctement un END à la fin du programme, avant le début des lignes de sous-programmes.

La *figure 6.5* montre une élaboration de la routine INKEY$? Le défaut d'INKEY$ est que l'instruction ne vous signale pas son usage : il n'y a pas de point d'interrogation d'affiché comme avec INPUT. Le sous-programme des lignes 1000 à 1040 porte remède à ce défaut en faisant clignoter un astérisque pendant que vous réfléchissez à votre choix. L'astérisque clignote grâce à l'affichage alterné de l'astérisque et de la séquence d'effacement. CHR$ (8) fait reculer le curseur d'une colonne, et CHR$ (16) efface le caractère à la place du curseur, sans faire avancer celui-ci.

Pour rendre le clignotement raisonnablement lent, j'ai ajouté un autre sous-programme, un délai à la ligne 2000.

```
10 CLS
20 PRINT"Choisissez 1 ou 2, S.V.P."
30 GOSUB 1000
40 PRINT "Votre choix était ";K$
50 END
1000 K$=INKEY$
1010 IF K$<>"" THEN RETURN
1020 PRINT "";:GOSUB 2000 RUN
1030 PRINT CHR$(8);CHR$(16);:GOSUB 2000
1040 GOTO 1000
2000 FOR J=1 TO 2000:NEXT:RETURN
```

Fig. 6.5. Un sous-programme d'astérique clignotant. L'astérisque clignote tant que vous n'appuyez sur aucune touche.

Essayez cette paire de sous-programmes dans vos programmes, et voyez la différence qu'ils apportent. Par la même occasion, familiarisez-vous avec les sous-programmes. Ils n'offrent pas seulement un moyen utile d'obtenir des opérations en plusieurs points d'un programme, ils fournissent une aide indispensable à l'organisation structurée des programmes, dont nous allons parler plus longuement.

Votre propre usage

Vous pouvez tirer beaucoup de plaisir de votre CPC464 en l'utilisant pour charger des programmes de cassettes achetées, ou de cartouches enfichables. Vous pouvez tirer encore plus de plaisir en tapant des programmes que vous avez trouvés dans les revues. Il est encore plus gratifiant de modifier l'un de ces programmes pour modifier son comportement, l'adapter à votre convenance. Le comble de la satisfaction, en ce qui concerne la programmation, est toutefois atteint quand vous concevez vos propres programmes. Il ne s'agit pas forcément de chefs-d'œuvre, il suffit de décider ce que vous voulez, l'écrire sous forme de programme, l'introduire et le faire fonctionner. C'est votre travail à cent pour cent, et vous en tirerez d'autant plus de satisfaction. Après tout, acheter un ordinateur sans le programmer par soi-même revient à acheter une Ferrari, pour la faire conduire par quelqu'un d'autre.

Maintenant, je ne peux deviner ce qui vous intéresse en programmation. Certains lecteurs peuvent vouloir concevoir des programmes pour garder des informations sur leur collection de timbres, de disques, de recettes de cuisine ou des détails techniques sur les vieilles locomotives à vapeur. Des programmes de ce type sont appelés des programmes de *bases de données*, parce qu'ils exigent l'entrée d'un grand nombre de données.

D'un autre côté, vous pouvez être intéressés par les jeux, les motifs colorés, les dessins, le son, et autres programmes qui font déplacer des formes sur l'écran. Des programmes de ce type requièrent des instructions que nous verrons en détail aux chapitres 8 à 10.

Nous allons examiner dans cette section un type de programme de bases de données, parce que sa structure peut être réutilisée dans toutes sortes de programmes. Une fois que vous savez organiser un programme simple de ce type, vous pouvez progresser en utilisant les mêmes méthodes pour concevoir vos propres programmes graphiques ou sonores.

Souvenez-vous toutefois que les programmes graphiques rapides ou élaborés que vous pouvez voir ne sont pas écrits en BASIC. La raison en est que le BASIC est trop lent à l'exécution pour permettre des mouvements rapides sur l'écran, ou le contrôle de plusieurs objets en même temps. Les programmes du type des jeux d'arcade sont écrits en *code machine*, un ensemble d'instructions codées numériquement que comprend directement le microprocesseur qui forme le cœur de l'ordinateur. Cela dépasse complètement le BASIC, et c'est beaucoup plus difficile. En fait, beaucoup de programmes en langage machine sont écrits à l'aide d'autres programmes, sur de très grosses machines, puis chargés sur les ordinateurs familiaux. Toutefois, si vous apprenez à concevoir des programmes en BASIC, vous serez capables d'apprendre plus tard le langage machine. Tout ce dont vous avez besoin est de l'expérience — beaucoup d'expérience.

Deux points sont importants ici. Le premier est que l'expérience compte pour la tâche de la conception de programme. Si vous choisissez des objectifs simples pour vos premiers projets, vous en tirerez bien plus d'enseignements. C'est parce que vous réussirez sans doute plus facilement à mener

à bien un programme simple, pour votre premier essai. Vous apprendrez plus en réalisant un programme simple qui fonctionne après un peu de réflexion et quelques modifications qu'en travaillant à un programme élaboré qui ne fait jamais exactement ce que vous attendez de lui.

Le deuxième point est que la conception de programmes doit commencer *avec l'ordinateur éteint,* et de préférence dans une autre pièce ! La raison en est que la conception de programme nécessite une planification, et que vous ne pouvez faire un plan correct quand vous avez la tentation d'un clavier devant nous. Éloignez-vous !

Écrivez !

Nous commençons donc avec un bloc de papier. J'utilise des feuilles perforées de format A4, de manière à pouvoir réunir les feuillets dans un classeur. De cette manière, je peux garder des feuillets en ordre, en ajouter à volonté aussi bien qu'en jeter, ce qui est tout aussi important. Oui, des feuilles ! Même un programme simple exigera très probablement plus d'une feuille de papier pour sa conception. Si vous en venez à des programmes plus élaborés, vous vous trouverez facilement aux prises avec un plan sur deux douzaines de feuilles, avec une liste de tâches avant de vous mettre au clavier.

Pour rendre l'exercice plus intéressant, je prendrai un exemple concret, et nous l'élaborerons au fur et à mesure. Ce sera un programme très simple, mais il illustrera toutes les compétences nécessaires.

Commencez donc par écrire ce que vous attendez du programme. Vous croyez peut-être que vous n'en avez pas besoin, parce que vous savez ce que vous voulez, mais vous pourriez avoir des surprises. Le vieux dicton sur l'arbre qui cache la forêt s'applique dans toute sa force à la programmation. Si vous n'écrivez pas ce que vous attendez d'un programme, il est à parier qu'il ne le fera jamais ! Vous allez vous retrouver perdu dans les détails quand vous allez commencer à écrire des lignes de BASIC, au point qu'il vous sera facile d'oublier votre premier objectif. Mais si vous écrivez votre objectif, vous aurez un but devant vous, et c'est aussi important en programmation que dans la vie.

Ne vous contentez pas de griffonner quelques mots. Passez-y du temps, et pesez ce que vous voulez faire faire au programme. Si vous ne le savez pas, vous ne pourrez le programmer ! Et l'avantage le plus important que vous en retirerez est que en écrivant vos objectifs, vous aurez une chance de concevoir un programme correctement *structuré*. Structuré signifie dans ce cas que le programme est composé en séquence logique, de sorte qu'il est facile de le compléter, le modifier ou le réorganiser. Si vous apprenez à programmer ainsi, vos programmes seront aisés à comprendre, seront plus vite mis au point, seront plus faciles à compléter pour qu'ils effectuent des tâches pour lesquelles ils n'étaient pas prévus au départ.

Prenez pour exemple la *figure 6.6.* Elle montre l'ébauche du programme d'un jeu simple. Le but du jeu est de se familiariser avec les noms d'animaux et de leurs petits. Le plan du programme montre ce que j'attends de ce jeu. Il doit présenter le nom d'un animal, pris au hasard, l'afficher et demander le nom de son petit.

Objectifs

1 Présenter le nom d'un animal sur l'écran.
2. Demander comment son petit s'appelle.
3. La réponse doit être correctement orthographiée.
4. L'utilisateur ne doit pas pouvoir lire la réponse sur le listing.
5. Donner un point pour chaque réponse correcte.
6. Donner deux chances à chaque question.
7. Garder une trace du nombre de tentatives.
8. Présenter le score comme le nombre de réussites par rapport au nombre de tentatives.
9. Choisir les noms d'animaux au hasard.

Fig. 6.6. L'ébauche du plan d'un programme. C'est le premier pas.

Un peu plus de réflexion permet d'ajouter quelques points. Le nom du petit devra être correctement orthographié. Un peu d'astuce sera nécessaire pour empêcher l'utilisateur (fils, fille, frère ou sœur) de trouver les réponses en tapant LIST pour chercher les lignes de DATA. Tout jeu doit avoir un système de score, aussi accordons-nous un point par bonne réponse. Puisque l'orthographe est importante, nous devons peut-être autoriser plus d'un essai à chaque question. Enfin,

nous devrions garder trace du nombre d'essais et du nombre de bonnes réponses, et présenter cela comme le bilan de chaque partie.

C'est à peu près tout ce dont nous avons besoin, à moins de vouloir un jeu plus élaboré. Pour un premier effort, c'est bien assez. Comment passons-nous à la conception à ce stade ?

La réponse est qu'il faut ébaucher le programme comme un peintre au début d'un tableau, ou un architecte au début d'un projet. Il faut donc tracer d'abord les grandes lignes avant d'en venir aux détails. Les grandes lignes d'un programme sont les opérations qui organisent la séquence des actions, ce qu'on appelle le *module d'enchaînement*. Par exemple, nous allons vouloir afficher un titre, donner à l'utilisateur le temps de lire avant de lui exposer les instructions de jeu. Il y a peu de doute que nous voudrons faire des choses comme assigner des variables, dimensionner des tableaux, et faire quelques autres préparatifs. Nous avons ensuite à faire le jeu proprement dit. Puis il faut trouver le score, et proposer à l'utilisateur une autre partie. Oui, il faut mettre tout cela par écrit ! La *figure 6.7* montre à quoi votre feuille peut ressembler à ce stade.

---

1. Afficher le titre, puis les instructions.
2. Afficher le nom de l'animal sur l'écran.
3. Demander le nom du jeune.
4. Utiliser INPUT pour la réponse.
5. Comparer la réponse avec la réponse correcte.
6. Si correct, ajouter 1 au score et demander si une autre question est désirée.
7. Si incorrect, autoriser un autre essai.
8. Si encore incorrect, choisir une autre question.
9. Fin si l'utilisateur tape N en réponse à « Encore une question ? »

Fig. 6.7. Le stade suivant dans l'élaboration de l'ébauche.

Les pierres de fondation

Maintenant nous pouvons enfin nous mettre à écrire un bout de programme. Ce sera juste une fondation, cependant. Vous devez éviter à tout prix de remplir des pages de BASIC à ce stade. Comme n'importe quel entrepreneur vous le dira, les

fondations comptent énormément. Faites-les bien, et vous aurez assuré les bases de la solidité du reste de la structure. Le principal est d'éviter de construire un mur avant d'avoir fait les fondations !

```
10 CLS:GOSUB 1000
11 REM TITRE
20 GOSUB 1200
21 REM Instructions
30 GOSUB 1400
31 REM Mise en Place
40 GOSUB 2000
41 REM JEU
50 GOSUB 3000
51 REM Score
60 GOSUB 4000
61 REM Encore ?
70 IF INSTR("Oui   ",k$)<>0 THEN 40
100 END
```

Fig. 6.8. Un *noyau* ou *fondation* ou *module d'enchaînement* pour l'exemple.

La *figure 6.8* montre quel devrait être votre objectif à ce stade. Il y a seulement quatorze lignes de programme ici, et c'est tout ce dont vous avez besoin. C'est une *base*, souvenez-vous-en, pas la tour Maine-Montparnasse. C'est aussi un programme en cours d'élaboration, aussi avons-nous accroché quelques panneaux « Danger travaux » ici et là. Ces avertissements prennent la forme de lignes qui commencent par REM. REM signifie remarque, et toute ligne de programme qui commence par REM sera ignorée par l'ordinateur à l'exécution. Cela signifie que vous pouvez taper n'importe quoi derrière REM, et que cela vous permet d'annoter votre programme. Ces notes ne seront pas affichées à l'exécution, vous ne les verrez que quand vous utilisez LIST. A la *figure 6.8*, j'ai mis les lignes de remarques sur des lignes dont le numéro est supérieur d'une unité aux lignes commentées. De cette manière, je pourrai retirer toutes les lignes de remarques ultérieurement. Quand ? Quand le programme sera achevé, testé et fonctionnera parfaitement.

Les remarques sont utiles, mais elles encombrent la mémoire, et ralentissent un peu le programme. J'aime bien

garder une copie de mes programmes avec les remarques, et un exemplaire « de travail » sans les remarques. Ainsi, j'ai un programme rapide et efficace pour l'usage quotidien, et une version complètement expliquée que je peux utiliser pour faire des modifications.

Revenons au programme lui-même. Comme vous pouvez le voir, il consiste en un ensemble d'instructions GOSUB, qui font référence à des lignes que nous n'avons pas encore écrites. C'est intentionnel. Ce que nous voulons à ce point, ce sont des fondations. Le programme suit exactement le plan de la *figure 6.7*, et la seule partie qui ne soit pas consacrée à un GOSUB est le IF de la ligne 70. Nous allons écrire un sous-programme qui utilisera INKEY$ pour regarder si O ou o a été tapé, et la ligne 70 traite la réponse. Quelle est la question ? Eh bien, c'est l'étape « Encore une partie ? » que nous avions prévue.

Oui, vous avez raison, la ligne 70 paraît assez originale. En testant avec INSTR(" OUIoui ",K$), on aura 1 si on appuie sur O, ou 4 si on appuie sur o. On obtiendrait aussi ces réponses si K$ contenait OUI ou oui (ce qui est impossible avec INKEY$).

Si K$ ne contient ni O ni o, INSTR renvoie 0, ce qui veut dire que la chaîne que nous examinons ne se trouve pas dans OUIoui. Simple, mais très utile.

Regardez bien attentivement ces quatorze lignes de programme, elles sont importantes. L'utilisation de tous ces sous-programmes signifie que nous pouvons facilement vérifier le programme — il n'y a guère de sources possibles d'erreurs dedans. Nous pouvons maintenant décider de l'ordre dans lequel nous allons écrire les sous-programmes.

Comme d'habitude, le mauvais ordre serait de suivre l'ordre dans lequel les appels à ces sous-programmes se succèdent. Écrivez toujours le titre et les instructions en dernier, parce que c'est le point qui vous importe le moins pour l'instant. En tout cas, si vous écrivez ces parties trop tôt, il y a des chances pour que vous deviez en modifier les instructions plus tard, une fois que vous aurez eu de brillantes idées pour améliorer le jeu !

Ce serait une bonne idée que d'écrire une ligne comme :

```
9 GOTO 30
```

pour sauter le titre et les instructions. Cela épargne beaucoup de temps quand vous testez le programme, puisque vous évitez le temps consacré à l'affichage du titre et des instructions à chaque exécution.

L'étape suivante consiste à passer au clavier (enfin !) et entrer ce noyau de programme. Si vous utilisez le GOTO pour dépasser titre et instructions temporairement, vous pouvez vous contenter de mettre de simples PRINT sur les lignes des sous-programmes correspondants. C'est ce que nous avons déjà fait au programme de la *figure 6.2* ; vous voyez donc comment procéder. Cela vous permet de tester votre noyau, et de vérifier qu'il fonctionnera avant d'aller plus loin.

Ensuite, il faut enregistrer ce noyau, et puis l'enrichir régulièrement. Si vous avez enregistré le noyau, vous pouvez le recharger dans votre CPC464, lui ajouter un sous-programme, vérifier le fonctionnement du résultat. Une fois satisfait, vous pouvez enregistrer le tout sur une autre cassette. La prochaine fois que vous voudrez ajouter un sous-programme, vous recommencerez avec cette cassette, et ainsi de suite.

De cette manière, vous gardez des bandes d'un programme qui s'accroît régulièrement, testé à chaque stade, et dont le bon fonctionnement est reconnu. A nouveau, cette procédure est très importante. Tester prend très souvent plus longtemps que vous ne le prévoyez, et cela peut être très fastidieux quand vous avez à faire à un long programme. En testant chaque sous-programme au fur et à mesure, vous avez confiance dans les éléments précédents, et vous pouvez vous concentrer sur les erreurs des nouvelles sections.

La routine des sous-programmes

Ensuite il faut concevoir les sous-programmes. En fait, certains n'exigent pas grande réflexion. Prenez par exemple le sous-programme qui doit être placé à la ligne 4000. C'est simplement la routine INKEY$ qui nous est familière, avec quelques PRINT, aussi pouvons-nous la traiter d'emblée. La *figure 6.9* montre la forme qu'elle pourrait prendre. Le sous-programme est sans détours, ce qui nous autorise à nous en débarrasser tout de suite ! Tapez-le, et testez le noyau avec ce sous-programme en place.

```
4000 PRINT"Aimeriez-vous une autre parti
e ?"
4010 PRINT"Répondez par O ou N, S.V.P."
4020 K$=INKEY$:IF K$="" THEN 4020
4030 RETURN
```

Fig. 6.9. Le sous-programme de la ligne 4000.

Nous en venons maintenant à ce que vous pourriez croire le plus difficile, le sous-programme qui effectue les opérations du jeu lui-même. En fait, vous n'avez aucune nouveauté à apprendre pour cette partie. Le sous-programme de jeu est conçu exactement comme le programme noyau. Cela veut dire que nous devons écrire ce que nous attendons de lui, et organiser les étapes qui mèneront à bien sa tâche. Si quoi que ce soit paraît mériter réflexion, nous pouvons le repousser dans un sous-programme auquel nous nous intéresserons plus tard.

1. Garder les réponses dans un tableau de chaînes de codes ASCII.
2. Garder la liste des animaux dans une autre chaîne.
3. Le nombre qui sélectionne l'animal sélectionnera aussi la réponse.
4. Utiliser la variable NT pour le nombre de tentatives, SC pour score.
5. Utiliser la variable NEQ pour enregistrer le nombre d'essais à une question.

Fig. 6.10. Plan du sous-programme de jeu.

Prenez comme exemple la *figure 6.10.* C'est le plan du sous-programme de jeu, qui inclut aussi l'information dont nous aurons besoin pour les étapes de mise au point. Le premier item est le résultat d'un peu de réflexion. Nous voulions, souvenez-vous, éviter qu'un utilisateur astucieux ne triche en regardant les réponses dans les lignes de DATA. La parade la plus simple est de mettre les réponses sous forme de code ASCII. Cela n'effrayera pas les plus habiles, mais sera suffisant pour des débutants. J'ai décidé de mettre une réponse dans chaque ligne de DATA sous la forme d'une chaîne de codes ASCII, où chaque code est écrit sous la forme d'un code à trois chiffres. Pourquoi trois ? Eh bien, parce que les capitales utilisent deux chiffres seulement, et les minuscules trois, aussi utiliser trois simplifie les choses. Vous verrez

pourquoi plus tard — on écrit donc un nombre comme 86 sous la forme 086, etc. C'est le premier item du sous-programme.

Ensuite nous allons conserver les noms des animaux dans un tableau. Cela présente plusieurs avantages. L'un est que si nous faisons ainsi, il est très facile de tirer au sort un élément. L'autre est que cela facilite aussi la mise en relation des questions et des réponses exactes. Si les questions sont les items d'un tableau dont les indices vont de 1 à 10, nous pouvons placer les réponses dans des lignes de DATA (un ensemble de nombres par ligne) et les lire également dans un tableau de chaînes.

Le plan règle ensuite le problème des noms de variables que nous allons utiliser. Cela aide toujours d'utiliser des noms qui nous rappellent ce que les variables sont censées représenter. Dans ce cas, utiliser SC pour le score, et NT pour le nombre de tentatives semble assez clair. La troisième variable, NEQ, que nous utiliserons pour compter combien d'essais ont été faits, signifie donc nombre d'essais à une question. Enfin nous prenons Q$ comme nom du tableau qui contiendra les noms d'animaux.

Le jeu pour aujourd'hui

La *figure 6.11* montre le résultat de la mise en œuvre du plan de la *figure 6.10*. Les étapes consistent à prendre un nombre au hasard, l'utiliser pour afficher un nom d'animal, et enfin trouver la réponse. C'est tout, car la vérification de la réponse et le score sont pris en charge par un autre sous-programme.

```
2000 NEQ=0:V=INT(10*RND(1))+1
2010 CLS:PRINT "L'animal est - ";Q$(V)
2020 PRINT:PRINT "On appelle le jeune -
";
2030 INPUT X$:NT=NT+1
2040 GOSUB 5000
2041 REM Trouver la réponse correcte
2050 RETURN
```

Fig. 6.11. Les lignes de programme pour le sous-programme de jeu.

Essayez toujours de diviser autant que possible le programme, pour éviter d'avoir à écrire de longs morceaux d'affilée. En l'occurrence, j'ai dû faire appel à un autre sous-programme pour garder les choses assez claires.

Nous commençons le sous-programme à la ligne 2000 en « remettant à zéro une variable ». NEQ est mis à 0, pour s'assurer que cette variable a la valeur correcte chaque fois que ce sous-programme est appelé. La deuxième partie de la ligne 2000 prend ensuite un nombre au hasard, entre 1 et 10. Les lignes 2010 à 2030 sont évidentes. On affiche le nom de l'animal qui correspond au tirage, on demande la réponse, le nom du petit. La dernière section de la ligne 2030 compte le nombre de tentatives. C'est l'endroit logique où placer cette opération, puisque nous voulons faire le compte à chaque réponse.

Ensuite, il est temps de se défiler : je ne veux pas me mêler pour l'instant de la lecture des codes ASCII. Je laisse donc cela à un sous-programme, débutant en 5000, que j'écrirai plus tard. La remarque de la ligne 2041 me rappelle ce que ce nouveau sous-programme devra faire, et le sous-programme de jeu se termine par le RETURN usuel.

Au milieu des détails

Une fois la routine de jeu en sécurité sur bande, on peut réfléchir aux détails. Le premier à examiner devrait précéder ou suivre le sous-programme de jeu. J'ai choisi la routine de score. Comme d'habitude, il faut en faire le plan, selon la *figure 6.12.* Chaque fois qu'il y a une réponse correcte, la variable numérique SC doit être incrémentée, et l'on peut revenir au programme principal.
Il faut aller plus loin si la réponse est inexacte. Il faut afficher

---

1. Pour une réponse correcte, incrémenter SC.
2. Pour une réponse incorrecte, avec NEQ=0, autoriser un deuxième essai, et mettre NEQ à 1.
3. Pour une deuxième réponse incorrecte, avec NEQ=1, passer à la question suivante, et remettre NEQ à 0.

Fig. 6.12. Le plan du sous-programme de Score.

un message, et offrir une autre chance. Si le résultat à ce nouvel essai est incorrect, il faut en finir. Plus tard, après avoir lu le chapitre 11, vous pourrez vouloir inclure des sons. On pourrait annoncer une faute par un bip bref, et une bonne réponse par un long. Écrivez cela vous-même!

```
3000 PRINT:IF X$=A$ THEN SC=SC+1:GOTO 32
00
3010 IF NEQ=0 THEN GOTO 3300
3020 NEQ=0:PRINT "Pas de chance. Essayez
 le suivant."
3030 FOR Q=1 TO 1000:NEXT
3040 RETURN
3200 PRINT "Correct. Votre score est dor
énavant";SC
3210 PRINT "en"NT;" tentative(s)."':GOSUB
 7000:GOTO 3040
3300 PRINT "Inexact - Mais cela vient de
 votre "
3310 PRINT "orthographe! Vous avez droit
 à un deuxième essai.":NT=NT-1
3320 GOSUB 7000:NEQ=1:GOSUB 2010:GOTO 30
00
```

Fig. 6.13. Le sous-programme de Score.

La *figure 6.13* montre le sous-programme qui développe ce plan. La ligne 3000 traite des réponses correctes. Il nous faut un message à cet endroit, et pour avoir un peu plus d'espace, on utilise GOTO 3200 pour terminer le travail. Le GOTO 3040 de la ligne 3210 assure que si la réponse est correcte on évite le reste du sous-programme, avant le RETURN. Si la réponse est incorrecte, la ligne 3010 entre en action. Elle teste la valeur de NEQ, et si c'est 0, elle envoie à la ligne 3300 pour afficher le message et donner de nouvelles instructions. La ligne 3320 appelle le sous-programme de la ligne 2010 à nouveau pour que l'utilisateur puisse proposer une nouvelle réponse. Le GOTO 3000 de la fin de la ligne 3320 teste la nouvelle réponse.

Il y a une astuce ici. La variable numérique NEQ commence avec pour valeur 0. Quand il y a une réponse incorrecte et que NEQ est encore à 0, la condition de la ligne est satisfaite. Toutefois, une des actions de la ligne 3320 est de mettre NEQ à 1.

Quand vous répondez au deuxième essai, avec NEQ = 1, et que votre réponse est fausse, la ligne 3010 ne vous déroute pas vers 3300, parce que NEQ est différent de zéro. La ligne suivante est donc effectuée, elle remet NEQ à 0 pour le mot suivant, affiche un message de condoléances, fait une pause et laisse le sous-programme revenir à la ligne 3040.

```
1400 NT=0:SC=0:NEQ=0
1410 DIM Q$(10),A$(10)
1420 FOR J=1 TO 10:READ Q$(J):NEXT
1430 FOR J=1 TO 10:READ A$(J):NEXT
1440 RETURN
```

Fig. 6.14. Le sous-programme de dimensionnement des tableaux.

Maintenant que nous tenons fermement le principal, nous pouvons polir les autres sous-programmes. La *figure 6.14* montre le sous-programme qui dimensionne les tableaux. La ligne 1400 met toutes les variables de score à zéro. La ligne 1410 dimensionne les tableaux Q$, qui contiendra les noms d'animaux, et A$, qui recevra les chaînes de nombres des réponses. La ligne 1420 remplit le tableau Q$ en lisant les noms d'une liste de DATA, et la ligne 1430 en fait autant pour les réponses avec A$. Et voilà ! On peut écrire les lignes de DATA plus tard, comme d'habitude.

```
5000 A$="":FOR J=1 TO LEN(A$(V)) STEP 3
5010 A$=A$+CHR$(VAL(MID$(A$(V),J,3))):NE
XT
5020 RETURN
```

Fig. 6.15. Vérification de la réponse.

Vient ensuite la tâche de trouver la réponse. Nous avons préparé un plan à cet effet, aussi cela ne devrait pas faire trop de pagaille. La *figure 6.15* montre les lignes de programme. La variable V est celle que nous avons prise au hasard, et elle sert à sélectionner l'une des chaînes de codes ASCII, A$(V).

Puisque chaque nombre comprend trois chiffres, nous voulons extraire trois chiffres à la fois de la chaîne, aussi utili-

sons-nous STEP 3 dans la boucle FOR...NEXT de la ligne 5000. La ligne 5010 construit ensuite la chaîne de la réponse, que nous appelons A$.

Souvenez-vous que A$ utilisé seul ne se confond pas avec les éléments de l'ensemble A$(V). A$ est mis à zéro (chaîne vide) au début de la ligne 5000 pour faire en sorte que nous commencions toujours avec une chaîne vide, en éliminant la réponse précédente qui se trouverait aussi dans A$. La chaîne A$ est ensuite construite en prenant trois chiffres, en les convertissant par VAL en nombre, puis en caractère par CHR$. Ce caractère est ensuite ajouté à A$, et cela continue jusqu'à l'épuisement des nombres de la chaîne. C'est la fin de l'épreuve.

La *figure 6.16* montre le sous-programme des instructions, et la *figure 6.17* celle du titre. Les lignes du titre incluent une pause, et elles ont été écrites en mode 0. Cela donne facilement de grandes lettres, et c'est excellent, tant que vous évitez les mots trop longs qui débordent de l'écran ! Finalement, la *figure 6.18* montre les lignes de DATA et le sous-programme de pause.

```
1200 CLS:PRINT TAB(15)"INSTRUCTIONS"
1210 PRINT:PRINT TAB(4)"L'ordinateur va
vous proposer le nom"
1220 PRINT"d'un animal. Vous devrez tape
r le nom de"
1230 PRINT "son petit - sans faute d'ort
hographe"
1240 PRINT "et commençant par une capita
le"
1250 PRINT "L'ordinateur gardera votre s
core."
1260 PRINT "Vous avez deux essais par no
m."
1270 PRINT
1280 PRINT " Appuyez sur la barre d'espa
cement pour commencer."
1300 IF INKEY(47)=-1 THEN 1300 ELSE RETU
RN
```

Fig. 6.16. Les instructions — à toujours laisser pour la fin.

```
1000 MODE 0
1010 PRINT TAB(7)"Les Petits"
1020 GOSUB 7000:MODE 1:RETURN
```

Fig. 6.17. Les lignes de titre du programme.

```
6000 DATA Chien,Chat,Vache,Cheval,Poule,
Renard,Kangourou,Oie,Lion,Cochon
6001 DATA 067104105111116
6002 DATA 067097116111110
6003 DATA 086101097117
6004 DATA 080111117108097105110
6005 DATA 080111117115115105110
6006 DATA 082101110097114100101097117
6007 DATA 075097110103111117114111117
6008 DATA 079105115111110
6009 DATA 076105111110099101097117
6010 DATA 080111114099101108101116
7000 FOR Q=1 TO 3000:NEXT:RETURN
```

Fig. 6.18. Les lignes de DATA nécessaires, avec un sous-programme d'attente.

On peut maintenant tout combiner et procéder à un essai. Souvenez-vous que le CPC464 vous permet d'enregistrer des morceaux de programme et de les réunir ensuite par MERGE. Comme le programme a été conçu par blocs séparés, il vous est facile de le modifier. Vous pouvez utiliser des données différentes, par exemple. Vous pouvez en utiliser beaucoup plus, à condition de changer le contenu de DIM à la ligne 1410. Vous pouvez en faire un jeu de questions et réponses sur un thème tout à fait différent, en changeant simplement les données et les instructions. Vous pouvez créer des effets sonores plus intéressants, ou ajouter des effets graphiques.

Un des principaux défauts du programme est qu'un item déjà proposé peut être repris car c'est l'un des effets possibles de RND. Vous pouvez dominer ce problème en échangeant l'item qui vient d'être utilisé avec le dernier item de la liste (sauf si c'était celui-là), et en diminuant le nombre maximum de choix : par exemple, si vous tirez 5, après le jeu, échangez l'item 5 et le 10, et tirez l'item suivant parmi les neuf restants.

Cela veut dire que l'expression 10*RND(1)+1 deviendra D*RND(1)+1, avec D décrémenté à chaque bonne réponse, à partir de 10.

En fait, il y a beaucoup à faire pour rendre ce programme plus attrayant. Je l'ai utilisé pour vous montrer que vous pouviez dès maintenant concevoir un ensemble complet. Prenez-le comme une sorte de « jeu de construction » en BASIC, pour refaire ce qui vous plaît. Il vous donnera une idée de la satisfaction que vous pouvez tirer de la maîtrise du CPC464. Quand votre expérience croîtra, vous deviendrez capable de concevoir des programmes beaucoup plus longs et plus élaborés que celui-ci. A ce moment, vous songerez à compléter votre CPC464 avec une imprimante et un lecteur de disquettes. Même avant d'en arriver à ce stade, vous verrez combien il est utile de garder des enregistrements des sous-programmes utilitaires que vous puissiez ajouter à vos programmes à l'aide de MERGE.

Encore une remarque : supposez que votre programme ne fonctionne pas à votre gré ? Comment faire pour tirer au clair les problèmes ? Pour un bref aperçu, lisez l'*appendice A*, qui traite de l'édition et de la recherche d'erreurs.

7

Les fichiers de données sur cassettes

Le CPC464, à la différence de la plupart des autres micro-ordinateurs, est livré avec un système d'enregistrement des programmes et données incorporé. Alors que la plupart des autres ordinateurs utilisent des magnétophones à cassettes ordinaires, le CPC464 utilise son propre enregistreur-lecteur, qui a spécialement été conçu pour un usage informatique. Comme l'utilisation d'un tel système pour enregistrer des données est probablement tout à fait neuf pour les propriétaires de CPC464, même s'ils ont déjà utilisé un ordinateur, ce chapitre est consacré à l'usage de cassettes pour enregistrer des données. Le chapitre 1 a déjà traité de l'utilisation du magnétophone pour stocker et récupérer des programmes.

Les mots qui risquent d'intriguer le plus le possesseur de CPC464 sont *périphérique, canaux* ou *voies* d'*entrées/sorties* et *tampon*. Une fois que vous aurez saisi ce que signifient ces termes, et comment on les emploie, vous trouverez la manipulation du magnétophone beaucoup plus intéressante, et vous serez capable de faire bien plus avec votre CPC464. Commençons donc par expliquer ces mots.

Un *périphérique* est quelque chose qui envoie ou reçoit des données. Votre clavier est un périphérique d'entrée, parce que chaque fois que vous appuyez sur une touche, un ensemble de signaux électriques est envoyé à l'ordinateur. L'écran est un autre périphérique, de sortie, lui, parce que vous verrez apparaître à l'écran un signal à chaque fois que l'ordinateur envoie au téléviseur un ensemble de signaux électriques.

L'écran du CPC464 se comporte comme plusieurs périphériques, parce qu'il peut être divisé en plusieurs portions, qui reçoivent chacune des signaux. Plus tard, au chapitre 8, nous verrons comment diviser l'espace de l'écran à votre guise.

Certains périphériques peuvent effectuer les opérations d'entrée et de sortie. Une *console* est la combinaison d'un clavier et d'un écran, qui envoie et reçoit à la fois des données. Le magnétophone est aussi un périphérique utilisable dans les deux directions. Un lecteur de disquettes est, lui aussi, un périphérique d'entrée/sortie.

Nous avons parlé de signaux électriques passant d'un périphérique à un autre, ce qui est effectivement ce qui se passe. Il est bien plus utile de voir ce que représentent ces signaux. Chaque ensemble de signaux représente une unité de données appelée un *octet* (sur les ordinateurs familiaux), et les ordinateurs sont faits pour traiter des *données.*

Un octet est la quantité de mémoire requise pour stocker un caractère en code ASCII, ou le code d'une instruction BASIC. Les données peuvent être des nombres ou des caractères; c'est tout ce que l'ordinateur a à manipuler. Si vous avez un programme qui classe les noms de vos amis par dates d'anniversaires, le programme doit avoir des données : dans cet exemple, l'ensemble des noms et des jours d'anniversaire ou dates de naissance. Si vous avez un programme qui donne des recettes et des listes de courses à faire, les données sont les instructions, les noms des produits, et les quantités. Tout ordinateur conçu pour faire plus que de simples jeux doit être capable de sauvegarder et charger des données de ce genre, séparément du programme qui les engendre ou les utilise.

Il y a beaucoup à gagner avec cette approche. La mémoire du CPC464 est utilisée à bien des choses sur lesquelles vous n'avez aucun contrôle. Un très long programme pour réunir des données et ensuite les utiliser pourrait ne pas tenir dans votre ordinateur. Il est beaucoup plus sensé de faire un programme court pour réunir les données, en utilisant des lignes d'INPUT, et pour les enregistrer. Les données sont alors en sécurité contre tout bouleversement de la mémoire du CPC464 et un autre programme peut ensuite les utiliser. En tenant les deux programmes et les données séparés, vous

pouvez traiter bien plus d'information qu'il ne serait possible si vous deviez tout faire tenir dans la mémoire en même temps.

Qu'est-ce que tout cela a à voir avec des tampons, des canaux d'entrées/sorties et des périphériques ? Eh bien, les périphériques sont les parties de l'ordinateur qui donnent ou reçoivent les données, comme nous le savons. Les *canaux* sont les voies qui transportent les données. Réfléchissez à ce qui se passe quand vous utilisez votre CPC464. Quand vous appuyez sur une touche, quelque chose apparaît sur l'écran. Le clavier est un périphérique, l'écran un autre, et il y a un canal qui les relie. C'est juste une manière élégante de dire qu'il y a un chemin pour les données entre le clavier et l'écran.

Le point important, cependant, est que ces chemins ou canaux peuvent être contrôlés. Les contrôler signifie que nous pouvons les changer, en interrompre certains, en créer d'autres, à notre gré. Interrompre certains canaux serait évidemment déraisonnable, sauf pour des raisons précises. Normalement vous voulez voir à l'écran les mots que vous tapez au clavier. Cependant, si vous tapiez un mot de passe spécial, que vous désiriez cacher à toute personne qui regarde l'écran, il serait logique d'interrompre le canal qui connecte le clavier et l'écran.

La *figure 7.1* montre comment le faire. La ligne 10 vous demande d'entrer un mot de passe de quatre lettres. A la ligne 20, PRINT CHR$(21) déconnectera le canal qui relie le clavier et l'écran. Vous pouvez maintenant introduire votre

```
10 CLS:PRINT "DONNEZ UN MOT DE PASSE DE
4 LETTRES"
20 PRINT CHR$(21)
30 INPUT B$
40 PRINT CHR$(6)
50 IF LEN(B$)<>4 THEN PRINT"Incorrect -
Réessayez":GOTO 10
60 PRINT"Merci - mot de Passe reconnu."
70 PRINT "Tapez PRINT B$ Pour le voir!"
```

Fig. 7.1. Comment cacher une entrée en interrompant la sortie vers l'écran.

mot de passe, en utilisant l'INPUT de la ligne 30, mais rien n'apparaîtra sur l'écran ! La ligne 40 remet en état le canal, si bien que les messages peuvent à nouveau être affichés. Comme d'habitude, l'entrée est testée, si bien que vous pouvez réessayer, au cas où vous n'auriez pas tapé quatre lettres. Si vous voulez vérifier que vous avez bien introduit quelque chose, PRINT B$ vous le révélera.

Comme vous pouvez vous y attendre maintenant, il y a des canaux connectés à des périphériques dès que vous mettez sous tension. Il est évident, par exemple, qu'il y a une connection entre le clavier et l'écran. C'est normalement un canal numéroté 0, et vous pouvez forcer une instruction PRINT à suivre ce canal en mettant #0 derrière PRINT. Le dièse # est le signe américain utilisé pour signifier « numéro ».

La conception du CPC464 permet d'utiliser dix canaux. Parmi eux, trois (0, 8 et 9) sont réservés pour l'usage propre du CPC464. Le canal #0 contrôle normalement l'écran, le canal #8 l'imprimante, et le canal #9 le magnétophone. Cela laisse sept numéros de canaux pour votre usage, et dans ce chapitre et le suivant, nous verrons comment utiliser ces canaux de réserve. Ce chapitre concerne l'enregistrement de données sur cassettes, ce qui implique que vous sachiez deux choses. L'une est la manière de sélectionner un tampon, l'autre est la manière de relier ce tampon au magnétophone.

On sélectionne un *tampon* et on le relie au magnétophone en utilisant une instruction OPEN. Il y en a deux variantes : OPENOUT et OPENIN.

OPENOUT doit être suivi d'un nom de fichier, tout comme SAVE. A l'exécution, cette instruction mettra le magnétophone en état d'enregistrer les données fournies par l'ordinateur. L'instruction constitue une simple préparation — aucune donnée ne passe au moment de l'exécution d'OPENOUT ; c'est seulement la préparation des données à transmettre.

OPENIN doit aussi être suivi d'un nom de fichier, tout comme LOAD. A cause de l'usage du nom de fichier, seul le fichier correspondant sera sélectionné, et le magnétophone *fonctionnera* de manière à trouver ce nom de fichier sur la bande.

Techniques d'enregistrement de données

Le mot *fichier* est fréquent dans ce livre. Fichier signifie *toute collection de caractères qui forme un ensemble.* Les caractères d'un programme BASIC constituent un fichier, par exemple, parce que le programme ne pourra être exécuté s'il en manque. Un ensemble de noms et d'adresses en ASCII est un fichier parce qu'ils forment un groupe cohérent d'informations, tels nos amis, nos fournisseurs ou débiteurs. Un ensemble d'octets de code machine est un fichier, aussi bien qu'une collection de nombres utilisés dans un programme financier.

Dans ce chapitre, je vais entendre par fichier une *collection d'informations enregistrables sur cassette, et différente d'un programme.* Par exemple, si vous avez un programme qui traite votre budget personnel, vous aurez besoin d'un fichier de produits et de montants d'argent. Ce fichier est le résultat de votre programme, et conserve d'une fois sur l'autre les montants. Prenons un autre exemple : supposons que vous avez conçu un programme destiné à gérer votre collection de vieux 78 tours. Le programme vous demande d'introduire tout un tas d'informations au sujet de ces enregistrements. Cette information constitue un fichier, et à un stade du programme, vous aurez à l'enregistrer. Pourquoi ? Parce que si un programme de cet ordre doit vous rendre des services, il n'y aura pas assez d'espace, même dans la vaste mémoire du CPC464, pour contenir toute l'information à la fois. De plus, vous ne voudrez pas avoir à changer le programme à chaque fois que vous souhaiterez ajouter des items à la liste.

C'est le sujet que nous traitons dans ce chapitre : enregistrer l'information que le programme traite. On dit aussi mettre en fichier l'information.

Nous ne pouvons discuter de la mise en fichier sans rencontrer des mots constamment utilisés dans ce domaine. Les termes les plus importants sont ceux de *fiche* (ou *enregistrement*) et de *champ (figure 7.2).* Une fiche est un ensemble de faits à propos d'un item du fichier. Par exemple, si vous avez un fichier sur les locomotives à vapeur vous pourriez avoir une fiche pour chaque type de locomotive. A l'intérieur d'une fiche, vous pourriez avoir le nom du concepteur, la superficie du foyer, la pression de vapeur, la force de traction, et tout ce

FICHIER AMIS

ENREGISTREMENT 1

CHAMP 1	Nom 1
CHAMP 2	Adresse 1
CHAMP 3	Téléphone 1
CHAMP 4	Date d'anniversaire 1

ENREGISTREMENT 2

CHAMP 1	Nom 2
CHAMP 2	Adresse 2
CHAMP 3	Téléphone 2
CHAMP 4	Date d'anniversaire 2

ENREGISTREMENT 3

etc.

Fig. 7.2. Le sens d'*enregistrement* et de *champ.*

qui est en rapport. Chacune de ces données est un *champ,* un membre du groupe qui constitue une fiche. Votre fiche peut être par exemple celle des locomotives SCOTT de la classe 4-4-0. Tout élément d'information au sujet de la classe SCOTT est un champ, l'ensemble des champs forme la fiche, et la fiche de la classe SCOTT n'est qu'un enregistrement du fichier qui inclura aussi les Pacific Gresley, les locos 4-6-0 tous usages, etc.

On peut prendre pour autre exemple le fichier « motos anglaises ». Dans ce fichier, vous aurez une fiche BSA, une fiche AJS, une fiche Norton. Dans chaque fiche, vous aurez des champs. Ces champs concerneront la cylindrée, le nombre de cylindres, l'alésage et la course du piston, la suspension, la vitesse maximale, l'accélération et tout ce que vous voulez retenir. Faire des fichiers est amusant — si vous aimez organiser les choses.

Les fichiers sur cassettes

Dans ce livre, comme nous parlons du CPC464 et de son magnétophone, nous omettrons les méthodes d'archivage fondées sur les lignes de DATA dans un programme. C'est parce que garder les données sur cassette séparément du programme est beaucoup plus utile. Même si vous êtes familier du procédé, restez patiemment avec moi, jusqu'à ce que j'aborde des sujets qui vous soient neufs.

Pour commencer, il y a deux types de fichiers, dont un seul est utilisable avec un magnétophone à cassettes. Ce sont les *fichiers séquentiels* et les *fichiers à accès direct*. La différence est simple, mais importante.

Un fichier séquentiel enregistre toutes les informations dans l'ordre sur une cassette. Si vous voulez atteindre un item, vous devez lire tous les enregistrements dans l'ordinateur, et puis choisir. Il n'y a pas de moyen simple de faire lire au système un seul enregistrement, ou un seul champ.

Un fichier à accès direct fait ce que son nom suggère, il vous permet d'extraire des données enregistrées dans une fiche ou un champ sélectionné sans lire tous les autres depuis le début du fichier. La différence entre fichier séquentiel sur bande et fichier à accès direct sur disque est illustrée sur la *figure 7.3.* Les fichiers à accès direct exigent l'usage de lecteurs de dis-

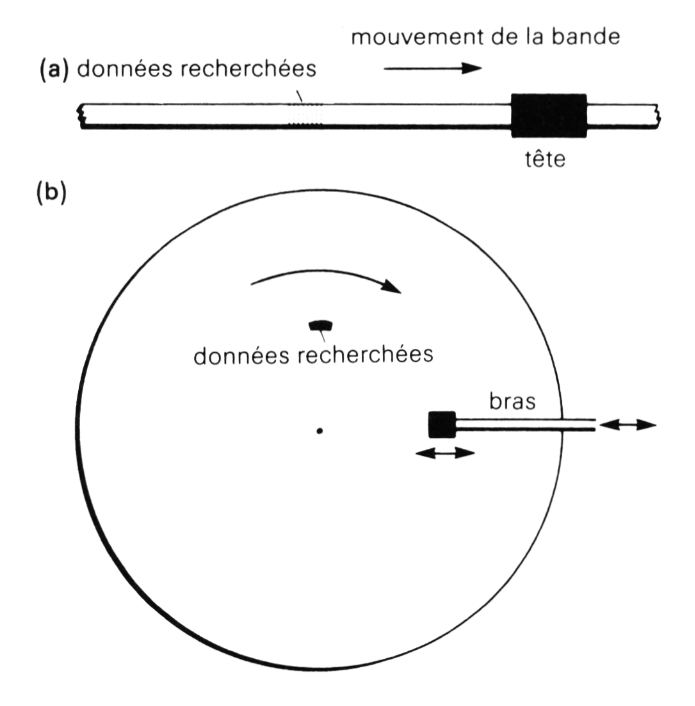


Fig. 7.3. Illustration de la différence entre (a) un fichier séquentiel sur bande, et (b) un fichier à accès direct sur disque. N'importe quelle partie du disque peut être atteinte en déplaçant la tête de lecture à une distance correcte pendant la rotation du disque. Pour lire un segment de bande, on doit faire passer toute la bande précédente devant la tête.

quettes, mais on peut concevoir des programmes qui ont un effet similaire en utilisant des fichiers séquentiels du magnétophone. Nous commencerons donc en examinant les fichiers séquentiels, qui correspondent au type de fichiers enregistrés sur cassette.

Création d'un fichier

Quand on aborde une nouveauté, il est toujours bon de partir du début, et de laisser tout d'abord les choses compliquées de côté. Nous examinerons les procédures de mise en fichier en regardant d'abord comment on fait la liaison avec le magnétophone. La *figure 7.4* montre un exemple très simple : comment enregistrer une donnée, la valeur de la variable appelée A, sur le magnétophone, dont le canal porte le numéro 9 dans tous les cas. Comme les différentes étapes sont importantes, nous allons les examiner en détail. A la ligne 10, d'abord, nous *ouvrons un fichier.* Cette opération utilise l'instruction OPENOUT, et nous devons spécifier le nom de ce fichier de données. La ligne 10 utilise OPENOUT « TEST » pour ouvrir un nouveau fichier sur le magnétophone, du nom de TEST.

```
10 OPENOUT "TEST"
20 A=5
30 PRINT#9,A
40 CLOSEOUT
```

Fig. 7.4. Enregistrement de la valeur d'une seule variable. C'est la *valeur* qui est enregistrée, et non le nom de variable.

Faites attention à la manipulation de l'enregistreur à ce point : vous devez choisir une cassette et une position sur la cassette sans rien d'enregistré. En effet il n'y a aucune protection contre l'effacement d'un fichier, qui peut se produire en enregistrant par accident sur lui un autre fichier, qu'il porte le même nom ou pas. Les systèmes de disquettes peuvent vous protéger contre cette sorte d'erreur, mais quand vous utilisez une cassette pour stocker des données, votre seule protection est la lecture du compteur de défilement.

L'étape suivante consiste en l'assignation d'une valeur à la

variable A, à la ligne 20. La ligne 30 est la plus importante, en fait. L'instruction PRINT #9, A signifie « envoyer la valeur de A sur le canal 9 ». Et comme le canal 9 est toujours connecté au magnétophone, la ligne va « afficher » la valeur de A sur la cassette. Vous verrez les messages habituels que vous obtenez quand vous utilisez SAVE, et la valeur sera enregistrée en fichier sous le nom TEST, qui permettra de la retrouver facilement. La ligne 30 effectue l'enregistrement, mais ce n'est pas aussi direct que cela paraît.

Le magnétophone enregistre des groupes de caractères, et le système qui le contrôle (on dit *système d'exploitation*) est conçu de manière à regrouper dans la mémoire de l'ordinateur les données jusqu'à ce qu'il y en ait assez pour former un groupe. La partie de la mémoire où se fait ce regroupement s'appelle un *tampon*, et quand vous ouvrez un canal, vous allouez aussi automatiquement un tampon à ce canal. Les données qui doivent être enregistrées sont transférées une à une sur le tampon, et puis enregistrées en bloc. S'il y a plus de données que le tampon peut en contenir (plus de 2000 caractères), le tampon devra être rempli et vidé plusieurs fois. A la fin d'un pareil processus, vous devez vous assurer que le tampon est supprimé. C'est fait à la ligne 40, par l'instruction CLOSEOUT.

Maintenant qu'est-ce qui se passe quand ce programme s'exécute ? En ce qui vous concerne, le magnétophone envoie simplement les messages usuels, tourne assez longtemps, puis s'arrête. Mais nous devons prouver que la donnée de notre exemple a bien été enregistrée, et montrer que nous pouvons la récupérer.

```
10 OPENIN "TEST"
20 INPUT#9,X
30 PRINT "X a pour valeur";X
40 CLOSEIN
```

Fig. 7.5. Récupération de la valeur d'une variable à partir de la bande.

Regardez maintenant le listing de la *figure 7.5.* La ligne 10 de ce listing utilise OPENIN, et non OPENOUT. Nous avons déjà un fichier sur la cassette, et nous voulons le relire, pas en

créer un nouveau. Le numéro de canal est à nouveau # 9, et nous devons spécifier « TEST », le nom du fichier. Après avoir « ouvert le fichier » (ce qui veut dire qu'un tampon est désormais alloué aux signaux provenant du magnétophone) le système trouve le nom de fichier sur la cassette, s'il existe. Si le nom de fichier ne peut être trouvé, le magnétophone continuera à lire la bande jusqu'au bout ! Un lecteur de disquettes effectue ces recherches bien plus efficacement, car il peut trouver tout de suite si le nom de fichier se trouve sur le disque.

Après avoir trouvé le nom de fichier, on peut lire les données. C'est le rôle de la ligne 20, qui utilise INPUT #9,X. INPUT seul envoie toujours au clavier, mais quand on place #9 après INPUT, cela fait que l'entrée est prise sur le magnétophone. Comme nous avons spécifié « TEST » dans l'instruction OPENIN, ce qui vient du magnétophone ne peut être que le contenu du fichier TEST. La ligne 30 affiche ce qui est lu, et la ligne 40 referme le canal.

Tout cela paraît assez simple et direct, mais examinez de près ces deux petits programmes, car ils contiennent beaucoup de points dont vous devez avoir la maîtrise pour la mise en fichier. Remarquez, par exemple, que l'on peut assigner ce qui est lu à n'importe quel nom de variable. Nous avons utilisé A à l'enregistrement, et X à la lecture. En ce qui concerne l'ordinateur, lire un fichier sur une cassette est une opération d'INPUT analogue à une lecture du clavier.

Essayez maintenant quelque chose de plus ambitieux avec la création d'un fichier de nombre. La *figure 7.6* montre un programme qui engendre un fichier de nombres — les nombres pairs de 0 à 50 — et enregistre ensuite ces nombres sur le magnétophone, tout en les affichant à l'écran. Il y a seulement six lignes dans ce programme, mais trois d'entre elles contiennent des instructions importantes que vous devez maîtriser. Nous commençons, logiquement, par la ligne 10. C'est l'une de ces instructions OPENOUT qui connectent un tampon et un canal. Le canal est le # 9, celui du magnétophone, et le nom du fichier est PAIRS. Le point suivant est de créer un fichier, ce qui est effectué par une boucle qui commence à la ligne 20. Celle-ci alloue une variable N à chacun des nombres pairs à son tour, avec le NEXT à la ligne 50. Comment plaçons-nous les nombres sur le tampon ? La

ligne 30 s'en charge, avec PRINT #9,N, et comme il y a une suite de nombres à enregistrer, le tampon accepte les différents nombres *avant* la mise en route de l'enregistrement. Le magnétophone n'a rien de la rapidité de l'ordinateur. La boucle FOR...NEXT des lignes 20 à 50 est facilement accomplie avant que le moteur du magnétophone ne démarre ! Chaque fois que la ligne 30 est exécutée, la valeur de N est stockée momentanément sur le tampon.

```
10 OPENOUT "PAIRS"
20 CLS:FOR N=0 TO 50 STEP 2
30 PRINT#9, N
40 PRINT#0,N;" ";
50 NEXT
60 CLOSEOUT
```

Fig. 7.6. Création d'un fichier de nombres et enregistrement des valeurs.

Ce terme de tampon est bien choisi, puisque l'action du tampon est de relier l'ordinateur et le magnétophone de manière à ce qu'ils travaillent ensemble sans à-coup. Dans cet exemple simple, tous les nombres engendrés sont simplement passés au tampon. Rien n'est enregistré alors ; le magnétophone ne se met pas à tourner. L'enregistrement des données se produit quand tous les nombres ont été rassemblés. CLOSEOUT signifie fermer le canal, et quand on ferme un canal, une partie de l'opération consiste à vider tout tampon lié au canal. De plus une marque *fin de fichier* est enregistrée, de façon que le système puisse identifier la fin du fichier même si plusieurs blocs de données ont été enregistrés.

Et voilà ! si vous exécutez maintenant ce programme, vous verrez les nombres apparaître sur l'écran, matérialisant l'exécution complète de la boucle. Vous obtiendrez le message vous demandant d'appuyer sur REC et PLAY seulement après la fin de la boucle.

Modification des messages automatiques d'écran

Encore un seul point avant de nous mettre sérieusement à cette histoire de données. Dans certaines applications, vous n'avez pas vraiment besoin des messages sur les touches du

magnétophone, et l'enregistrement des blocs. Vous pouvez supprimer ces messages en mettant devant le premier caractère du nom du fichier un point d'exclamation. Par exemple, si vous modifiez la ligne 10 en OPENOUT « !PAIRS », à l'exécution du programme vous verrez les nombres, et l'enregistrement commencera sans rien de plus à l'écran. C'est utile, à condition que vous n'oubliiez pas d'appuyer sur REC et PLAY! Mais cela vous donne l'occasion de contrôler les choses par vous-même, en mettant vos propres messages, comme nous le verrons.

```
10 CLS:OPENOUT "!AUTRES PAIRS"
20 PRINT "Appuyez maintenant sur REC et
PLAY"
30 PRINT "Tapez un espace pour commencer
."
40 WHILE INKEY(47)=-1:WEND
50 CLS:FOR N=0 TO 5000 STEP 2
60 PRINT#9, N
70 PRINT N
80 NEXT
90 CLOSEOUT
100 "Fin de l'enregistrement. Appuyez su
r stop, S.V.P."
```

Fig. 7.7. Un fichier de nombres beaucoup plus long, pour illustrer l'action du tampon. Les messages ont été supprimés en utilisant le symbole ! dans le nom du fichier.

Ce programme vous a montré le tampon dans son rôle d'intermédiaire, et la taille du tampon est assez grande pour recevoir beaucoup de données. Vous pouvez le vérifier en modifiant le programme selon le modèle de la *figure 7.7.* Ce programme utilise le point d'exclamation dans le nom du fichier pour supprimer les messages, si bien que les lignes 20 et 30 fournissent un message approprié pour vous donner le loisir de préparer le magnétophone. Cette fois le nombre d'octets de données dépassera plusieurs milliers, et quand vous exécuterez ce programme, vous verrez que les nombres apparaissent sur l'écran jusqu'à 598. A ce point, le moteur du magnétophone se met en marche, et un bloc de données est enregistré. Les nombres se remettent à courir jusqu'à 1160, la cassette enregistre un autre bloc, puis on va jusqu'à 1672, et

ainsi de suite, tous les 512 nombres. En fait le tampon est rempli par la boucle, puis vidé par le magnétophone avant de recevoir un nouvel ensemble de données.

Ces courts programmes ont mis des données dans un fichier, mais jusqu'ici vous devez me croire sur parole quant à la présence des informations sur la bande. La *figure 7.8* vous montre comment faire dans le cas du fichier « PAIRS ». Dans cet exemple, le nom de fichier !PAIRS a été utilisé pour supprimer les messages usuels. Nous devons donc fournir les nôtres, mais comme vous le verrez sur le programme, vous devez faire attention à l'emplacement que vous leur donnez. Si vous commencez le programme par OPENIN « !PAIRS », vous constaterez que le programme ne fonctionne pas correctement. C'est parce que, à l'exécution d'OPENIN « !PAIRS » la machine doit trouver le nom de fichier « PAIRS » (même si celui-ci n'a pas été enregistré comme !PAIRS) avant de faire quoi que ce soit d'autre.

```
10 CLS:PRINT " Appuyez sur PLAY, Puis su
r la barre d'espacement"
20 WHILE INKEY(47)=-1:WEND
30 OPENIN "!Pairs"
40 FOR N=0 TO 50 STEP 2
50 INPUT#9, A
60 PRINT A;" ";
70 NEXT
80 CLOSEIN
90 PRINT:PRINT "Appuyez sur la touche ST
OP du lecteur de cassette."
```

Fig. 7.8. Lecture du fichier PARIS, sans les messages. Cela implique que vous fournissiez vos propres messages.

Vous devez donc mettre votre message « APPUYEZ SUR PLAY » tout au début du programme, pour que OPENIN puisse jouer son rôle. Comme d'habitude le programme commence par une ligne 30 qui ouvre le fichier, avec OPENIN « !PAIRS ». On lit le fichier avec INPUT #9, dans une boucle qui lit et affiche les données. Quand cela s'exécute, vous entendez le moteur du magnétophone se mettre en marche, la machine localise le fichier et le lit, et vous voyez les nombres apparaître à l'écran.

Essayons maintenant le fichier plus long de la *figure 7.9*, !AUTRESPAIRS. Cette fois, nous allons arranger un affichage plus propre en interrompant le processus de sortie à l'écran. La ligne 70 effectue l'interruption.

```
10 CLS:PRINT "Appuyez sur la touche PLAY
 du lecteur de cassette."
20 PRINT "Puis sur la barre d'espacement
 pour commencer."
30 OPENIN "!AUTRES PAIRS"
40 FOR N=0 TO 5000
50 INPUT#9, K
60 PRINT K
70 IF N/20=INT(N/20) THEN FOR J=1 TO 200
0:NEXT:CLS
80 NEXT
90 PRINT "Appuyez sur la touche STOP du
lecteur de cassette."
```

Fig. 7.9. Lecture du fichier AUTRESPAIRS, avec un affichage qui laisse le temps de lire les nombres.

La condition est IF N/20=INT(N/20). Cela signifie comme condition que N soit un multiple de 20. Si N/20 a un reste, N/20 n'est pas égal à la partie entière de N/20, qu'on obtient par INT(N/20). A chaque fois que N est un multiple de 20, la deuxième partie de la ligne 70 cause une pause, et à la fin de la pause, l'écran est nettoyé avant la suite de la boucle principale.

Quand vous exécuterez ce programme, vous entendrez le démarrage du moteur de la cassette, pour chercher le fichier et le lire. Ensuite le nombre 0 apparaît sous le texte de l'écran, et la boucle principale affiche les nombres par groupe de vingt. Le moteur du magnétophone démarrera et s'arrêtera par intervalles, pour remplir le tampon. Vous entendrez ces arrêts et départs tout le temps que les données sont relues dans le fichier AUTRESPAIRS. L'opération est longue, et il vaut mieux enregistrer les données à la vitesse rapide, en utilisant SPEED WRITE 1. Mais si les données ont été enregistrées à la vitesse lente, vous devez les lire aussi à la vitesse lente.

Incidemment, vous remarquerez que CLOSEIN a été omis dans cet exemple. Vous pouvez vous en tirer sans dommages

si vous exécutez le programme, et puis utilisez NEW pour remettre à zéro la machine. Si vous deviez lire d'autres données, vous ne pourriez utiliser un autre OPENIN sans avoir au préalable exécuté CLOSEIN.

Encore des fichiers séquentiels

Supposons que nous voulions enregistrer non un ensemble de nombres engendré par un programme, mais un ensemble de noms tapés au clavier. En ce qui concerne le magnétophone, ce n'est qu'un autre ensemble de données, et c'est traité exactement de la même manière. Chaque fois que vous tapez sur ENTER dans une instruction INPUT, les données sont stockées sur un tampon, et y restent tant que celui-ci n'est pas plein, ou qu'on n'a pas mis fin à la saisie et fermé le fichier. Encore une fois, vous voyez l'importance de l'usage du tampon : vous n'espérez pas que le magnétophone enregistre chaque caractère au moment où vous le tapez ?...

La *figure 7.10* montre un court programme de ce type. Normalement, si vous réunissiez de l'information ainsi, vous stockeriez les noms dans un tableau. Comme vous le savez sans doute, cela introduit des complications, telle la nécessité de dimensionner le tableau. A moins de vouloir consulter une entrée précédente en cours de saisie, vous n'avez pas besoin d'utiliser un tableau pour enregistrer un ensemble de noms. L'affaire se présente sans doute différemment à la relecture, mais nous y viendrons bientôt.

Pour entrer dans les détails de programmation, la ligne 10 choisit la vitesse SPEED WRITE 1, qui écrit plus vite. Cela *ne* signifie *pas* que la cassette défile plus vite, mais que les données sont enregistrées dessus à un rythme plus grand. Nous annulerons cette instruction à la fin du programme, sans quoi elle resterait active. Il se peut que vous vouliez sauver par SAVE un autre item, à vitesse lente, et cela ne serait pas possible après l'exécution de SPEED WRITE 1. La ligne 20 ouvre le fichier, en supprimant les messages comme d'habitude. Les lignes 30 à 60 donnent de brèves instructions, et la ligne 70 fournit une pause.

La boucle principale WHILE commence ensuite à la ligne 80 ; l'idée est de saisir par INPUT et d'enregistrer chaque nom, jusqu'à ce qu'on rencontre X ou x comme réponse au clavier.

```
10 SPEED WRITE 1
20 OPENOUT "!NOMS"
30 CLS:PRINT TAB(18)"NOMS"
40 PRINT:PRINT "  Ce Programme enregistr
e un fichier de noms sur une cassette.":
PRINT " Assurez-vous que vous avez":PRIN
T "une cassette Prete."
50 PRINT "Entrez X Pour arreter l'enregi
strement."
60 PRINT "  Appuyez sur REC et PLAY,  Pu
is sur la barre d'espacement Pour commen
cer quand vous etes Prets."
70 WHILE INKEY(47)=-1:WEND
80 WHILE NOM$<>"X" AND NOM$<>"x"
90 INPUT "NOM";NOM$
100 PRINT#9, NOM$
110 WEND
120 CLOSEOUT
130 PRINT "Appuyez sur STOP maintenant."
140 SPEED WRITE 0
```

Fig. 7.10. Fichier de noms. On a sélectionné la vitesse d'enregistrement rapide pour améliorer l'efficacité du programme.

Si vous vouliez rendre cet exemple plus fonctionnel, vous souhaiteriez probablement des instructions plus détaillées. La ligne 90 accepte le nom que vous tapez (sans virgule, dans un INPUT, souvenez-vous-en). Le nom est ensuite enregistré à la ligne 100. Si x n'a pas été tapé, la ligne 110 renvoie à la saisie du nom suivant. Si vous tapez des noms rapidement et sans interruption, vous constaterez que le moteur du magnétophone se met à tourner de temps en temps, pour vider le tampon. Vous ne pouvez introduire de nouvelles données pendant ce temps.

Encore de la lecture de données

Dorénavant, vous avez quelques fichiers de nombres et de noms stockés sur vos cassettes, et il est temps de faire plus attention aux méthodes de lecture de fichier, et à l'usage des données qui s'y trouvent.

Supposez par exemple que les nombres que nous avons enregistrés dans le fichier PAIRS ont été placés dans le fichier par un programme de comptabilité. Ils pourraient constituer par exemple les recettes quotidiennes d'un petit magasin. Vous pourriez entre autres opérations vouloir lire ces nombres sur le fichier, et les additionner, pour ne montrer que le total. C'est un problème de programmation classique, sans détours, pour lequel vous trouverez probablement beaucoup d'autres usages.

```
10 CLS
20 PRINT TAB(18)"TOTAUX"
30 PRINT:PRINT"Appuyez sur la touche PLA
Y du magnétophone":PRINT "Quand la casse
tte est prete."
40 PRINT " Appuyez sur la barre d'espace
ment pour commencer la lecture."
50 WHILE INKEY(47)=-1:WEND
60 OPENIN "!PAIRS"
70 TOTAL=0
80 FOR N=1 TO 26
90 INPUT#9, J
100 TOTAL=TOTAL+J
110 NEXT
120 PRINT:PRINT "Le total est";TOTAL
130 CLOSEIN
```

Fig. 7.11. Relecture du fichier PAIRS à l'aide d'une boucle FOR...NEXT.

La *figure 7.11* montre ce qui est nécessaire. On commence par nettoyer l'écran en ligne 10, la ligne 20 affiche le titre, et des instructions sont données aux lignes 30 et 40. La ligne 50 arrête la machine tant que l'on n'a pas tapé d'espace, et le vrai travail commence à la ligne 60 ; on ouvre le fichier !PAIRS. Maintenant, quand nous avions enregistré le fichier PAIRS, nous avions choisi les nombres pairs de 0 à 50, ce qui fait 26 nombres. Pour relire le même ensemble de nombres, la ligne 80 utilise une boucle à 26 passages. Le total a été mis à zéro à la ligne 70. La ligne 90 saisit chaque nombre sur le fichier, l'attribue à la variable J, et la ligne 100 ajoute ce nombre au total. A la fin de la boucle, la ligne 120 affiche la valeur du total, et le fichier est fermé à la ligne 130.

Supposez que vous ne connaissiez pas le nombre d'items enregistrés ? Cela rend l'usage de FOR...NEXT impossible, faute de connaître le nombre de passages nécessaires dans la boucle. En l'occurrence, le problème est facilement réglé. Le système de mise en fichier du CPC464 met un code de fin de fichier à la fin du dernier bloc de données qu'il enregistre. Et ce code de fin de fichier peut être détecté par l'instruction EOF.

```
10 CLS
20 PRINT TAB(18)"TOTAUX"
30 PRINT:PRINT"Appuyez sur la touche PLA
Y du magnétophone":PRINT "Quand la casse
tte est prete."
40 PRINT " Appuyez sur la barre d'espace
ment pour commencer la lecture."
50 WHILE INKEY(47)=-1:WEND
60 OPENIN "!PAIRS"
70 TOTAL=0
80 WHILE EOF=0
90 INPUT#9, J
100 TOTAL=TOTAL+J
110 WEND
120 PRINT:PRINT "Le total est";TOTAL
130 CLOSEIN
140 PRINT "Appuyez sur la touche STOP du
 magnétophone."
```

Fig. 7.12. Une meilleure méthode pour relire un fichier. Cette fois, vous n'avez pas à savoir combien il y a d'enregistrements.

Si EOF=0, la fin du fichier n'est pas encore atteinte, si EOF=1, elle est atteinte. Notre programme peut alors être commodément écrit comme à la *figure 7.12.* Cette fois nous utilisons une boucle WHILE...WEND à la place de FOR...NEXT. La condition de WHILE est EOF=0, car tant que EOF=0, on n'a pas atteint la fin des données de ce fichier.

Donner les noms

Maintenant que nous avons relu et utilisé un fichier de nombres, il est temps de voir la relecture du fichier de noms que

nous avons créé précédemment. Quand ce fichier a été créé, chaque nom a été enregistré avant que la marque usuelle de fin de fichier ne soit automatiquement mise sur la bande. Normalement, nous souhaitons placer les noms dans un tableau de façon que l'ordinateur puisse utiliser les données. Utiliser un tableau nous permet d'effectuer des tâches comme celle de classer alphabétiquement les noms, par exemple.

Tous les usages n'impliquent pas forcément l'usage d'un tableau. Supposez par exemple que vous vouliez chercher les noms commençant par un J. Vous pourriez le faire en utilisant le programme de la *figure 7.13.*

```
10 CLS:PRINT TAB(13)"RECHERCHE DE NOM":P
RINT
20 PRINT "Ce programme vous trouvera un
nom ":PRINT "parmi ceux du fichier noms.
"
30 PRINT " Appuyez sur PLAY quand la cas
sette est prete.":PRINT " Appuyez sur la
 barre d'espacement pour commencer."
40 WHILE INKEY(47)=-1:WEND
50 SPEED WRITE 1
60 INPUT "Première lettre du nom cherché
";Q$
70 OPENIN "!NOMS"
80 WHILE EOF=0 AND Q$<>LEFT$(N$,1)
90 INPUT#9, N$
100 WEND
110 PRINT:PRINT "Le nom est ";N$
120 CLOSEIN:SPEED WRITE 0
130 PRINT "Appuyez sur la touche STOP du
 magnétophone."
```

Fig. 7.13. Recherche d'un item dans un fichier. Ici, recherche d'un nom qui commence par une lettre donnée.

Les quelques premières lignes suivent le modèle habituel. Quand le programme est exécuté, il permet de trouver un nom dans le fichier en tapant simplement son initiale. Vous devez choisir SPEED WRITE 1 à la ligne 50 parce que le fichier NOMS a été enregistré à vitesse haute. Quand la vitesse rapide a été utilisée, vous devez vous assurer que la

cassette a été complètement rembobinée jusqu'au début du fichier.

La vitesse rapide est moins tolérante, et supporte mal de relire une zone où des informations ont déjà été enregistrées ! La ligne 60 vous demande l'initiale d'un nom (n'oubliez pas la capitale !). La ligne 70 ouvre le fichier en lecture et la ligne 80 commence une boucle qui prend un nom sur le tampon, vérifie d'abord si la fin du fichier est atteinte, et puis si la première lettre du nom est identique à l'initiale choisie. Si tout le fichier n'a pas encore été lu, cette partie compare la première lettre que vous avez choisie avec la première lettre du nom.

Si vous avez parcouru toute la liste sans trouver d'initiale correspondante, la ligne 80 mettra fin au programme quand elle trouvera la marque de fin de fichier. Si, d'un autre côté, le nom que vous voulez est trouvé, la ligne 110 l'affichera, et la recherche s'arrêtera là aussi. La boucle WHILE...WEND est un moyen très économique de programmer ce genre de recherche.

Ce programme fonctionne très bien pour une petite quantité de données, mais seulement si toutes les données sont différentes. Si vous avez deux noms comme Marie et Marguerite, la recherche de nom commençant par M vous donnera seulement le premier rencontré dans le fichier. Vous devrez modifier les tests dans la boucle pour obtenir que le programme affiche tout nom correspondant au critère de choix.

Beaucoup de programmes de ce genre sont traités plus efficacement en utilisant un tableau pour stocker les données. Le magnétophone n'est alors utilisé que comme un magasin et toute la sélection se fait dans la mémoire de l'ordinateur. Vous pourriez vous demander dans ce cas si cela présente un avantage par rapport aux lignes de DATA. C'est avantageux parce que les données peuvent être créées par un programme, et exploitées par plusieurs autres. Vous pouvez faire du programme qui lit et utilise les données un programme assez court, de sorte qu'il y ait beaucoup de place pour de nombreuses données dans la large mémoire du CPC464. Le magnétophone, en d'autres termes, vous permet d'utiliser des programmes courts avec beaucoup de données.

Stockage d'un fichier dans un tableau

La *figure 7.14* vous montre comment relire les données d'un fichier et les introduire dans un tableau. Le fait de les introduire dans un tableau nous pose un problème : il faut dimensionner le tableau pour qu'il contienne tous les items, donc nous devons en connaître le nombre.

```
10 CLS:PRINT TAB(13)"RECHERCHE DE NOM"
20 PRINT:PRINT TAB(2)" Chargez le fichie
r de noms suivant les instructions."
30 PRINT "Quand le fichier est completem
ent chargé":PRINT "  tapez la première l
ettre des noms que vous voulez voir."
40 PRINT " Taper 0 pour arreter le proce
ssus."
50 PRINT:PRINT "rembobinez complétement
la cassette noms":PRINT "Mettez le compt
eur à zéro."
60 PRINT " Appuyez sur PLAY, puis taper
un espace quand vous etes pret."
70 WHILE INKEY(47)=-1:WEND
80 PRINT:PRINT TAB(10)"Attendez S.V.P...
"
90 SPEED WRITE 1
100 OPENIN "!NOMS"
110 J=0:WHILE EOF=0
120 INPUT#9, N$
130 J=J+1
140 WEND:CLOSEIN
150 CLS:PRINT "Appuyez sur STOP."
160 PRINT:PRINT"Rembobinez à nouveau la
cassette et appuyez sur":PRINT "la touch
e PLAY, puis tapez un espace"
170 PRINT "quand vous etes pret."
180 WHILE INKEY(47)=-1:WEND
190 PRINT:PRINT TAB(10)"Attendez S.V.P..
."
200 OPENIN "!NOMS"
210 DIM NOM$(J)
220 FOR N=1 TO J
230 INPUT#9, NOM$(N)
240 NEXT:CLOSEIN:SPEED WRITE 0
250 PRINT:PRINT "Appuyez sur STOP mainte
nant."
260 FOR X=1 TO 2000:NEXT
```

```
270 CLS:PRINT:PRINT "Taper la première l
ettre du nom, S.V.P.":Q$="X":M=_
280 WHILE Q$<>"0"
290 INPUT Q$:Q$=LEFT$(Q$,1):M=0
300 FOR X=1 TO J
310 IF Q$=LEFT$(NOM$(X),1) THEN PRINT NO
M$(X):M=1
320 NEXT
330 IF M=0 AND Q$<>"0" THEN PRINT "Nom a
bsent. Essayez un autre."
340 WEND
350 PRINT "Fin du Programme"
```

Fig. 7.14. Transfert d'un fichier dans un tableau. Le fichier est lu une première fois pour trouver la dimension à donner au tableau, puis une deuxième pour saisir les items.

C'est assez facile si nous avons utilisé un tableau lors de l'enregistrement de ces items. Supposez par exemple que nous utilisions un tableau où mettre les noms lors des instructions INPUT, en déclarant un tableau NOMS$, et en le remplissant par des INPUT NOMS$(J) au lieu de procéder directement à l'enregistrement. Nous pourrions alors ouvrir le fichier, enregistrer la valeur de J avec un PRINT #9,J et puis faire une boucle pour enregistrer les éléments du tableau.

Mais si nous n'avons pas utilisé de tableau au départ, ni compté les éléments au moment de l'enregistrement, que pouvons-nous faire ? Une méthode consiste à garder en note le nombre d'éléments, et à fournir ce nombre en réponse à une question du programme de relecture. Par exemple on pourrait utiliser :

INPUT « Combien d'items » ; N :DIM NOMS$(N)

pour faire le dimensionnement. Une autre possibilité est d'utiliser un compteur dans le programme d'enregistrement comme :

INPUT NOMS$:N = N + 1

et d'enregistrer ce nombre en utilisant un nom de fichier qui relie ce nombre au programme principal, sur un autre morceau de bande. Par exemple si le programme principal est enregistré comme NOMSDAMIS, l'autre pourrait être NOMBREDAMIS.

Cela peut paraître lourd, mais c'est peu par rapport à l'avantage d'un dimensionnement précis. Dimensionner précisément veut dire que vous n'aurez jamais de message d'erreur parce que vous tentez de mettre trop d'éléments dans un tableau. Cela veut aussi dire que vous utilisez au mieux la mémoire du CPC464 et que vous faites la différence entre utiliser autant d'élements que nécessaire, et être restreint à bien moins.

Si malgré tout vous avez un fichier comme notre NOMS, dans lequel le nombre de noms n'est pas connu, peut-être parce que des noms ont été ajoutés lors d'une mise à jour, nous devons prendre des mesures radicales. Il va falloir compter les noms avant de nous en servir. C'est là encore un des avantages des fichiers sur disquettes par rapport aux bandes : un nombre peut être enregistré sur un disque avant ou après un ensemble de données, et on peut lire ce nombre avant les autres données si nécessaire.

La *figure 7.14* montre une solution à notre problème, qui ne fait pas perdre trop de temps, même pour des fichiers assez longs. Dans ce programme, tous les items sont lus un à un et comptés jusqu'à ce qu'on rencontre la fin de fichier. On vous demande ensuite de rembobiner la cassette. Maintenant qu'on connaît le nombre d'items, on peut dimensionner correctement le tableau et le fichier peut être relu, pour placer cette fois les items dans le tableau. Le choix des noms peut ensuite être fait dans une boucle, parce que le fichier n'a plus à être lu.

Le détail du programme est le suivant : les lignes 10 à 60 affichent les instructions, et la ligne 70 est le détecteur d'espace usuel. A la ligne 80, le message d'attente s'affiche pour vous rappeler que vous attendez le chargement des données. On sélectionne la vitesse rapide parce que l'enregistrement s'est fait ainsi, et on remet J à zéro. Dans la boucle on lit tous les items, et on incrémente J à chaque item. Il faut fermer le fichier à la ligne 140, parce qu'il faudra le rouvrir plus tard. Il faut ensuite rembobiner la cassette jusqu'au zéro, et les messages fournissent un rappel. On utilise ensuite la valeur de J pour dimensionner le tableau NOMS$ à la ligne 210, juste avant la lecture des noms du fichier. Comme on connaît le nombre de noms, on peut utiliser ici une boucle FOR...NEXT aux lignes 220 à 240 pour introduire les noms dans le tableau.

Une fois la lecture achevée, on peut mettre en œuvre les opérations de « RECHERCHE DE NOM » dans les lignes 280 à 350. Notez qu'il faut une valeur postiche pour Q$ avant le début de la boucle de recherche. Si un nom est trouvé, la variable M est mise à 1. Si un nom ne peut être trouvé avec la bonne initiale, M=0 et le message de la ligne 330 s'affiche. Quand vous appuyez sur 0 pour mettre fin à la recherche, cela met aussi M à 0, et le message de la ligne 330 est supprimé grâce à la condition IF M = 0 AND Q$ < > « O ». Si Q$ = « O », le message *n'*est *pas* affiché. C'est bien plus propre ainsi !

Quelques corrections : la modification d'un fichier

Soyons nets dès le départ, vous ne pouvez pas modifier un fichier enregistré sur magnétophone. Ce que vous pouvez faire, toutefois, est lire un fichier, le modifier, et le réenregistrer. Comme le CPC464 utilise un magnétophone intégré, vous pouvez profiter de ce trait pour enregistrer le nouveau fichier, en utilisant *le même nom*, sur une autre partie de la cassette ou sur une autre cassette.

Utiliser le même nom permet de lire le fichier mis à jour avec les mêmes programmes que pour l'ancien. Ce n'est pas aussi facile avec un système de disques. La technique illustrée à la *figure 7.15* suit de près le programme de la *figure 7.14*.

```
10 CLS:PRINT TAB(10)"MISE A JOUR"
20 PRINT:PRINT TAB(2)" Chargez les noms
suivant":PRINT" les instructions."
30 PRINT"Quand le fichier est completeme
nt chargé":PRINT"suivez les instructions
 pour":PRINT"le réenregistrer."
40 PRINT "Vous pouvez maintenant ajouter
 des items au fichier":PRINT"Tapez 0 pou
r arreter."
50 PRINT:PRINT "Rembobinez la cassette n
oms.":PRINT"Mettez le compteur à zéro."
60 PRINT " Appuyez sur PLAY, puis tapez
un espace quand vous etes pret."
70 WHILE INKEY(47)=-1:WEND
80 PRINT:PRINT TAB(10)"Attendez S.V.P...
."
90 SPEED WRITE 1
100 OPENIN "!NOMS"
```

```
110 J=0:WHILE EOF=0
120 INPUT#9, N$
130 J=J+1
140 WEND:CLOSEIN:J=J-1
150 CLS:PRINT:PRINT "Appuyez sur STOP."
160 PRINT:PRINT "Maintenant, rembobinez
la bande et":PRINT "appuyez sur PLAY. Ta
pez un espace à nouveau,"
170 PRINT "quand vous etes pret."
180 WHILE INKEY(47)=-1:WEND
190 PRINT:PRINT TAB(10)"Attendez S.V.P..
."
200 OPENIN "!NOMS"
210 DIM NOM$(J)
220 FOR N=1 TO J
230 INPUT#9, NOM$(N)
240 NEXT:CLOSEIN
250 PRINT:PRINT "Appuyez maintenant sur
STOP."
260 FOR X=1 TO 2000:NEXT
270 CLS:PRINT:PRINT TAB(5)"Préparez-vous
 à réenregistrer le fichier."
280 PRINT:PRINT"Trouvez un emplacement v
ierge sur la cassette,":PRINT "ou utilis
ez une cassette neuve."
290 "Remettez à zéro le compteur et suiv
ez"
300 "les instructions.":PRINT:PRINT "App
uyez sur REC et PLAY, puis tapez sur la
barre d'espacement."
310 WHILE INKEY(47)=-1:WEND
320 OPENOUT "!NOMS$"
330 FOR N=1 TO J
340 PRINT#9, NOM$(N):NEXT
350 CLS:PRINT:PRINT TAB(17)"AJOUTS"
360 PRINT:PRINT "Tapez les noms à ajoute
r au fichier":PRINT"maintenant. Tapez 0
pour arreter."
370 NOM$="X":WHILE NOM$<>"0"
380 INPUT NOM$
390 PRINT#9, NOM$
400 WEND
410 SPEED WRITE 0
420 CLOSEOUT
430 PRINT "Fin du programme."
```

Fig. 7.15. Mise à jour d'un fichier. Cela oblige à lire le fichier, faire les changements, puis réenregistrer sur une autre partie de la bande.

Les lignes 10 à 260 sont virtuellement les mêmes que celles de la *figure 7.14*, une fois mises à part les instructions. Ces lignes dimensionnent un tableau, puis le remplissent de noms. La ligne 140, toutefois, contient maintenant J=J—1. Cela évite d'utiliser le dernier item du tableau, qui est le terminateur x ou 0, qui a été utilisé dans le programme de création du fichier. Ainsi on a un fichier séquentiel avec un x ou un 0 à la fin et un seul, plutôt qu'un fichier où ces marques sont dispersées suivant les emplacements des différentes mises à jour.

Les lignes 270 à 340 préparent le réenregistrement du fichier. Je dis préparer parce que, sauf si le fichier de noms est très long, rien ne semble se passer aux lignes 320 à 340. C'est parce que le fichier est simplement envoyé au tampon, et non enregistré à ce stade. On vous demande de taper de nouveaux noms.

Tandis que vous le faites, le tampon peut se remplir, et demander à être vidé sur la cassette. Pendant cette opération vous devrez interrompre la frappe de nouveaux noms. Si vous ne tapez que quelques noms, le tampon n'arrivera pas à saturation et vous n'entendrez pas le moteur du magnétophone tant que vous n'aurez pas tapé le zéro qui arrête les entrées. Cette dernière opération met fin au nouveau fichier.

Si vous voulez le vérifier, tapez RUN à nouveau, et quand on vous demande de taper de nouveaux noms, tapez ESC deux fois, puis :

FOR Z=1 TO J:? NOM$(Z) ; « » ;:NEXT

et ENTER. Vous verrez alors tous les noms listés sur l'écran. Si vous voulez les voir mieux classés, ou sélectionnés suivant une lettre, ou en ordre alphabétique, vous devrez concevoir le programme par vous-même !

8

Fenêtres et autres effets

Il n'est jamais monotone de travailler avec un ordinateur, mais on peut faire beaucoup pour rendre le travail plus intéressant. Un des effets spéciaux qu'offrent les ordinateurs modernes est le *fenêtrage*. Dans ce chapitre, nous allons examiner cet effet ainsi que d'autres concernant l'affichage à l'écran. L'idée générale est de faire des affichages plus attrayants que du texte ou des schémas nus.

Les fenêtres

Une « fenêtre » signifie simplement une section d'écran qui peut être utilisée indépendamment du reste de l'écran, et même indépendamment de l'écran considéré comme un tout. Une fenêtre peut recevoir du texte, peut être déroulée ou nettoyée indépendamment de ce qui arrive aux autres sections.

Essayez ceci : remplissez l'écran avec un listing, ou n'importe quel texte ; tapez maintenant comme une commande directe (sans numéro de ligne) :

WINDOW 15,25,2,6 (et puis ENTER).

Maintenant utilisez CLS. Vous verrez que seul une petite partie de l'écran, la fenêtre, sera nettoyée. Essayez de taper un court programme dans cette fenêtre. Vous constaterez qu'elle se comporte comme si c'était le seul écran dont vous disposiez. Elle choisit automatiquement une nouvelle ligne quand c'est nécessaire, déroule le texte au besoin, tout comme le plein écran.

Pendant ce temps le reste de l'écran reste inaffecté. Un CLS nettoiera seulement la fenêtre, et non le reste de l'écran. Pour revenir rapidement à l'écran normal, tapez MODE 1 (ENTER).

Si cela a stimulé votre appétit, il est temps de voir comment contrôler cette histoire de fenêtres à notre guise. WINDOW doit être suivi de quatre nombres. Ces nombres sont analogues à ceux de LOCATE, et spécifient, dans l'ordre, les positions gauche, droite, sommet et base de la fenêtre.

Quand vous utilisez ce type d'instruction WINDOW, tout votre affichage se passe dans la fenêtre définie. Il utilise en fait le canal #0, qui est le canal normal pour PRINT et les autres instructions du même type. Vous pouvez cependant commencer les spécifications de WINDOW par un numéro de canal, comme # 2. Dans ce cas, la fenêtre n'est utilisée que lorsque vous avez une instruction comme PRINT # 2, “ FENETRE ”.

```
10 CLS
20 WINDOW#2,10,30,1,5
30 WINDOW#3,12,28,10,15
40 PRINT#2,"Fenetre #2 en usage"
50 GOSUB 150
60 PRINT#3,"Regardez le texte dans la fe
netre #2 -Fenetre #3 en usage"
70 GOSUB 150
80 CLS #2
90 GOSUB 150
100 CLS #3
110 GOSUB 150
120 PRINT#2,"Fenetre #2"
130 PRINT#3,"Fenetre #3"
140 END
150 FOR N=1 TO 2000:NEXT:RETURN
```

Fig. 8.1. Création d'une fenêtre sur le canal #2.

La *figure 8.1* montre ce type de fenêtre en action. Tout l'écran est nettoyé, et ensuite une fenêtre est reliée au canal # 2 par l'instruction :

WINDOW # 2,10,30,1,5.

Elle signifie que le canal numéro 2 est relié à une fenêtre d'écran. Les nombres qui suivent # 2 et la virgule spécifient la taille et la position de cette fenêtre. Son côté gauche sera à la colonne 10, et son côté droit à la colonne 30. Son sommet est à la ligne 1, et sa base à la ligne 5. L'effet de cette instruction est donc de vous permettre d'utiliser des instructions comme CLS #2 et PRINT #2 pour nettoyer cette fenêtre, ou y afficher.

Cette action est un choix permanent : à moins de fermer le canal ou de l'allouer à quelque chose d'autre, cette fenêtre restera contrôlée par l'utilisation du canal #2. Une autre grande différence est que l'ensemble de l'écran peut encore être nettoyé et recevoir de l'affichage. Nettoyer l'ensemble de l'écran nettoiera aussi la fenêtre, et afficher sur l'ensemble de l'écran recouvrira aussi la fenêtre, bien que vous puissiez toujours nettoyer à nouveau la fenêtre avec un CLS # 2.

Dans cet exemple, donc, la première fenêtre est créée à la ligne 20, et une autre fenêtre plus bas sur l'écran est créée à la ligne 30. Pour prouver l'existence de ces fenêtres, la ligne 40 imprime une phrase sur la fenêtre #2. Incidemment, vous pouvez introduire dans votre programme LIST #2 pour le lister sur cette fenêtre, mais la machine n'exécutera aucune ligne de programme qui pourrait suivre cette instruction. Ensuite il y a une pause causée par le sous-programme de la ligne 150. Après la pause, un nouveau texte est affiché sur l'autre fenêtre, #3. Le reste du programme illustre ensuite comment on peut nettoyer les fenêtres, et afficher dessus séparément. Remarquez toutefois, à la fin du programme, comment le message « Ready » et le curseur apparaissent au sommet gauche de l'écran. L'écran principal, dont le canal est le numéro 0, recouvrira toujours n'importe quelle fenêtre.

Échanges de fenêtres

On peut faire d'autres tours de fenêtrages en utilisant une autre instruction, WINDOW SWAP. Supposez, par exemple, que vous utilisiez un programme qui requière que vous tapiez des données, pour les enregistrer sur cassette. C'est le genre de cas que nous avons étudié au chapitre 7. Vous pourriez utiliser une fenêtre quelque part sur l'écran, au milieu, pour vos

entrées, et en réserver une autre pour les messages. Vous pourriez désirer par exemple que les messages usuels concernant le magnétophone apparaissent ailleurs. Si vous définissez une fenêtre près du bas de l'écran, en utilisant par exemple #3, vous pourriez effectuer :

WINDOW SWAP #0, #3

pour faire en sorte que tout ce qui va normalement à l'écran principal, comme les messages pour cette cassette, aille plutôt à la fenêtre #3.

```
10 CLS
20 WINDOW#2,2,39,5,10
30 WINDOW#3,2,39,25,25
40 PRINT#3,"Taper des noms (X pour arret
er)."
50 WHILE NOM$<>"X" AND NOM$<>"x"
60 INPUT NOM$
70 WEND
80 WINDOW SWAP 0,3
90 OPENOUT "NOMS"
100 PRINT#9,"NOM$"
110 CLOSEOUT
120 REM: Tapez ESC pour arreter
```

Fig. 8.2. Utilisation de WINDOW SWAP pour empêcher les messages d'apparaître en plein écran.

La *figure 8.2* montre cet effet (simulé). Les routines complètes pour l'enregistrement ont été omises, pour éviter de perdre du temps. Vous pouvez placer une cassette dans l'enregistreur, et utiliser PLAY seulement, quand on vous invite à enregistrer — cela vous évitera de gaspiller de la bande.

Il y a un autre moyen de faire ce genre de chose, incidemment. Vous constaterez que normalement vous ne pouvez pas appuyer sur la touche REC quand il n'y a pas de cassette dans la machine. Mais si vous poussez fort votre doigt contre le coin arrière gauche du logement de la cassette, vous constaterez que vous pouvez enclencher la touche REC. Il y a à cet endroit un petit levier utilisé pour détecter si une cassette est présente, et votre doigt peut le pousser suffisamment pour le tromper. C'est une astuce utile si vous voulez expérimenter

des programmes sans faire d'enregistrement effectif sur une cassette.

Pour en revenir au programme, les deux fenêtres qui sont définies ont les numéros #2 et #3. La fenêtre #3 n'a qu'une ligne, au bas de l'écran. Ainsi chaque message est vu séparément, sans confusion possible avec les messages précédents. Vous devez bien sûr vous assurer que vos messages ne dépassent jamais une ligne !

Le programme est sans détours jusqu'à la ligne 80. Cette ligne échange les fenêtre 0 et 3 — notez que vous n'avez pas à utiliser #0 et #3, et que vous obtiendrez un message d'erreur si vous le faites. Dès lors tous les messages qui apparaîtraient autrement au sommet de l'écran principal apparaîtront sur cette fenêtre. C'est une astuce très propre et utile pour séparer vos messages de votre autre texte.

Écrivez en couleur

Il est maintenant temps de regarder les instructions concernant la couleur sur le CPC464. Quatre sont importantes ; elles utilisent les instructions BORDER, PAPER, PEN et INK.

Nous commencerons par la plus simple, BORDER. BORDER peut être utilisé avec un nombre, ou deux. Si vous utilisez un numéro de couleur, vous aurez une bordure de couleur fixe au pourtour de votre écran principal. Vous avez la liste des numéros de couleurs à la *figure 8.3.* Si vous utilisez *deux* couleurs avec BORDER, le pourtour clignotera entre ces deux couleurs.

La *figure 8.4* vous en donne un avant-goût. Le programme commence par définir une fenêtre qui servira à afficher les numéros de couleur sur la droite de l'écran principal. Ensuite commence une boucle qui parcourra toutes les couleurs de BORDER disponibles. Chaque couleur est maintenue sur l'écran par une boucle de pause. A la différence des pauses précédentes, celle-ci est obtenue en utilisant l'horloge interne de la machine.

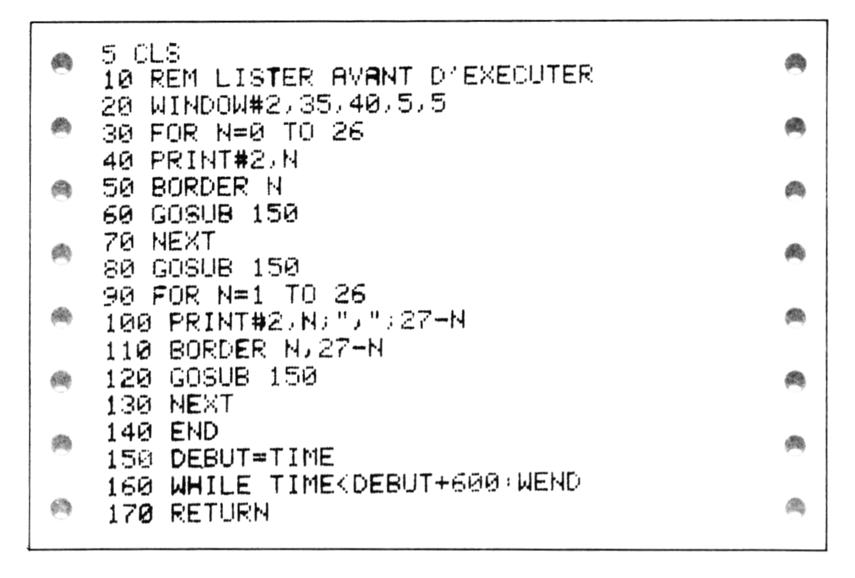
TIME est un nombre initialisé à zéro à l'allumage de la machine, et incrémenté 300 fois par seconde. Maintenant si vous mettez au point la routine des lignes 150 à 170, vous

Numéro	Couleur
0	Noir
1	Bleu
2	Bleuf vif
3	Rouge
4	Magenta (rouge pâle+bleu pâle)
5	Mauve
6	Rouge vif
7	Pourpre
8	Magenta vif
9	Vert
10	Cyan (bleu pâle+vert pâle), turquoise
11	Bleu ciel
12	Jaune
13	Blanc
14	Bleu pastel
15	Orange
16	Rose
17	Magenta pastel
18	Vert vif
19	Vert marin
20	Cyan vif
21	Vert citron
22	Vert pastel
23	Cyan pastel
24	Jaune vif (jaune d'or)
25	Jaune pastel
26	Blanc brillant

Fig. 8.3. Les numéros de couleur. Les nombres sont classés en ordre de luminosité croissante, tels qu'ils apparaîtraient sur un moniteur ou un téléviseur monochrome.

affectez à une variable la valeur de TIME à un instant donné. A chaque seconde écoulée, TIME aura augmenté de 300, aussi une boucle WHILE...WEND qui attend que TIME devienne DEBUT plus 600 entraînera une pause de 2 secondes. C'est simple, élégant, et plus facile à régler qu'une boucle FOR...NEXT.

Cependant, de retour au programme, après avoir parcouru les couleurs simples, on passe en revue les couleurs clignotantes. Le rythme de clignotement est fixé par une autre variable, et vous pouvez le modifier vous-même. Tandis que le pourtour continue à clignoter à la fin du programme, essayez de taper SPEED INK 10,10. Quand vous appuierez sur



```
5 CLS
10 REM LISTER AVANT D'EXECUTER
20 WINDOW#2,35,40,5,5
30 FOR N=0 TO 26
40 PRINT#2,N
50 BORDER N
60 GOSUB 150
70 NEXT
80 GOSUB 150
90 FOR N=1 TO 26
100 PRINT#2,N;",";27-N
110 BORDER N,27-N
120 GOSUB 150
130 NEXT
140 END
150 DEBUT=TIME
160 WHILE TIME<DEBUT+600:WEND
170 RETURN
```

Fig. 8.4. Affichage des couleurs de pourtour par BORDER, avec TIME pour créer des pauses.

ENTER, vous verrez bientôt le rythme de clignotement augmenter. Tapez maintenant SPEED INK 20,100, et regardez l'effet. Le premier nombre mesure le temps pour une couleur, le second pour la seconde. Les nombres représentent des cinquantièmes de seconde (ou des soixantièmes, suivant la fréquence du courant alternatif qui vous est fourni par le secteur). Ainsi un nombre de 50 devrait vous fournir des éclairs d'une seconde sur une couleur.
Attention à l'usage de cette instruction : *certains rythmes d'environ 8 par seconde peuvent être nuisibles pour les gens atteints d'épilepsie.*

En plus de son aspect désagréable, une bordure clignotante peut aussi être nuisible. Vous verrez que la taille de l'image de votre téléviseur ou moniteur change quand la bordure clignote. C'est parce que ces engins ne sont pas conçus pour prendre en compte des images clignotantes — un moniteur qui pourrait le faire serait trop cher pour la plupart d'entre nous (comptez environ 9000 F au cas où vous voudriez vous en offrir un !). Pour arrêter le clignotement, tapez BORDER 1 et ENTER. Un changement de MODE n'affecte pas l'état de BORDER.

Les items suivants sont PAPER, PEN et INK. Ces noms signifient en anglais papier, plume et encre respectivement. Leur sens n'est que partiellement en relation avec leur fonction. En particulier, si vous avez fait votre apprentissage sur un Spectrum, vous trouverez ces noms un peu gênants. Si ces termes sont complètement nouveaux pour vous, d'ailleurs, vous les trouverez aussi gênants — aussi suivez bien ! Le premier à examiner est INK.

INK signifie une couleur, et dans chaque mode, vous avez un nombre limité d'INKs disponibles. Le MODE 0 vous autorise jusqu'à 16 différentes couleurs d'INK sur l'écran. C'est-à-dire *n'importe quelle combinaison de seize couleurs choisies dans la liste de 27 couleurs possibles à la figure 8.3.* Le MODE 1, le mode texte normal, vous permet d'utiliser jusqu'à quatre couleurs d'INK. A nouveau, évidemment, vous pouvez choisir quelle combinaison de quatre parmi les 27 couleurs disponibles vous voulez utiliser. Le MODE 2, le mode à haute résolution, à 80 colonnes, vous permet deux couleurs d'INK seulement, choisies parmi les 27.

Une bonne manière de se représenter INK est de considérer que vous avez une palette avec un nombre limité de pots. En MODE 1, par exemple, vous avez quatre pots, quatre encriers. Vous pouvez mettre n'importe laquelle de vos vingt-sept couleurs dans chaque pot — mais vous ne pouvez peindre qu'avec les couleurs des pots. Si vous voulez utiliser d'autres couleurs, il vous faut remplir à nouveau vos pots !

Maintenant c'est là que le CPC464 commence à se montrer très différent d'autres machines. En MODE 1, vos *numéros* d'INK sont 0,1,2 et 3. J'ai dit *numéros* et non *couleurs* ici, parce que ces nombres *ne* sont *pas* des numéros de couleurs. Si nous revenons à notre comparaison avec une palette, ces numéros sont seulement les numéros des pots ou encriers. On peut utiliser l'instruction INK pour remplir les pots des couleurs choisies. La syntaxe de l'instruction est par exemple INK 2,7. Cette instruction remplira l'encrier 2, le pot 2, avec l'encre dont la couleur est le 7 (pourpre).

Quand vous commencez en MODE 1, les pots sont remplis pour vous. Le pot 0 est rempli de bleu, et c'est le pot qui décide de la couleur de votre fond d'écran. Le pot 1 est rempli de jaune d'or (en fait appelé vert brillant), couleur 18. C'est le

pot utilisé pour la couleur du texte. Le pot 2 est empli de couleur 20, cyan brillant, et le pot 3 de couleur 6, rouge brillant. La *figure 8.5* vous rappelle ces valeurs, qui sont utiles à connaître.

Mode 0 et Mode 1 :

Encrier 0	...	couleur 1	...	bleu
Encrier 1	...	couleur 24	...	jaune vif
Encrier 2	...	couleur 20	...	cyan vif
Encrier 3	...	couleur 6	...	rouge vif

(En Mode 2, seules les couleurs 1 et 24 sont utilisées initialement.)

Fig. 8.5. Les valeurs d'INK à l'initialisation.

Comment changer ces couleurs ? En utilisant l'instruction INK. En MODE 1, vous pouvez utiliser les couleurs d'INK de 0 à 3 (des nombres plus grands que 3 sont utilisables, mais vous donneront les mêmes couleurs). Pour allouer l'encrier 1 à la couleur 15, par exemple, vous n'avez qu'à taper : INK 1,15. N'oubliez pas l'espace entre le « K » d'INK et le nombre.

Si vous tapez *deux* numéros de couleurs, séparés par une virgule, votre pot contient de l'encre clignotante ! La couleur que vous utilisez clignotera entre les deux couleurs spécifiées. Par exemple INK 1,3,7 vous donnera une encre qui clignote entre les couleurs 3 et 7. Vous pouvez modifier le rythme de clignotement en utilisant SPEED INK comme précédemment. Le procédé rappelle les vieilles blagues sur la recherche de peinture rayée, ou à pois !

Quand vous mettez sous tension l'ordinateur, celui-ci vous met en MODE 1, et alloue les couleurs d'encrier « par défaut » selon ce que j'ai montré plus haut. Il décide aussi des encriers à utiliser pour le fond et pour le texte. Le fond, ou PAPER (papier), utilise le numéro d'INK 0, et le texte ou PEN (plume) utilise le numéro d'INK 1.

Maintenant je dois à nouveau insister, surtout si vous êtes un ancien utilisateur de Spectrum, sur le fait que ces numéros d'INK *ne sont pas des numéros de couleurs,* mais juste des numéros d'encriers. La couleur qui apparaît à l'écran dépend

de la couleur d'encre qui y a été mise. En utilisant l'instruction PAPER, vous pouvez modifier le numéro d'encrier utilisé pour le fond. En utilisant PEN, vous pouvez modifier le numéro d'encrier utilisé pour le texte.

```
10 BORDER 0
20 FOR N=0 TO 3
30 PAPER n:CLS
40 GOSUB 140
50 NEXT
55 PAPER o:CLS
60 FOR N=0 TO 3
70 PEN N
80 GOSUB 170
90 GOSUB 140
100 NEXT
110 GOSUB 140
120 PEN 1
130 END
140 DEBUT=TIME
150 WHILE TIME<DEBUT+600:WEND
160 RETURN
170 PRINT " Voici du texte pour montrer
l'effet de PEN";N
180 RETURN
```

Fig. 8.6. Utilisation des instructions PAPER et PEN.

La *figure 8.6* illustre l'usage de PAPER et PEN. Une instruction comme PAPER 2 ne suffit pas à faire apparaître une couleur, à elle seule. Si on affiche par PRINT sur l'écran après une instruction PAPER 2, le fond de notre affichage apparaîtra en cyan brillant, mais seulement pour la partie sur laquelle on a affiché. Pour donner la couleur de fond de PAPER 2 à tout l'écran, il faut utiliser après une instruction CLS. C'est illustré à la ligne 30. La seconde partie du programme utilise PAPER 0, qui est bleu foncé, parce que c'est la couleur de l'encre du pot numéro 0, et écrit en différentes couleurs de PEN.

Vous voyez d'après les résultats de ce programme que si vous voulez laisser votre texte bien lisible, vous devez utiliser des couleurs qui contrastent. Les couleurs dont la luminosité est

très proche ne donneront jamais assez de contraste pour faire un bon affichage. Ma propre préférence va à un papier sombre, et un texte clair parce que l'opposé, écran clair et encre sombre, peut fluctuer de manière assez fatigante. Ne vous attendez pas à voir des lettres de couleur très nette si vous utilisez un récepteur T.V., car les téléviseurs ne sont pas bien adaptés à l'affichage de très petits points colorés. Ajoutez à cela le fait que 9 % de la population mâle est partiellement daltonienne, et vous voyez que les affichages colorés les plus impressionnants sont ceux qui utilisent des couleurs vives, en larges zones, affichées sur un moniteur.

Quand vous exécuterez le programme, vous verrez que la ligne de texte pour PEN 0 n'apparaît pas. C'est parce que l'encrier 0 est celui utilisé pour le fond, le papier et la couleur. Si vous changez la couleur du fond avant d'écrire le texte, cependant, cette ligne apparaîtra, et l'une des autres sera invisible. J'ai utilisé le MODE 1 pour les exemples ici, parce que le choix de quatre encriers est commode. Souvenez-vous pourtant que vous avez le choix entre deux encriers en MODE 2, et entre 16, numérotés de 0 à 15, en MODE 0.

Les instructions PEN et PAPER peuvent être utilisées pour spécifier les couleurs utilisées dans les fenêtres. La *figure 8.7*

```
10 BORDER 0
20 WINDOW#1,1,40,3,5
30 WINDOW#2,1,40,7,10
40 WINDOW#3,1,19,11,24
50 WINDOW#4,20,40,11,24
60 A$=" Voici une phrase test pour mettr
e en valeur les fenetres."
70 PAPER#1,0
80 PAPER#2,1
90 PAPER#3,2
100 PAPER#4,3
110 PEN#1,3
120 PEN#2,2
130 PEN#3,1
140 PEN#4,0
150 CLS
160 FOR N=1 TO 4
170 PRINT#N,A$:NEXT
```

Fig. 8.7. PAPER et PEN utilisés dans des fenêtres différentes.

illustre cela, et montre aussi que toutes les combinaisons de couleurs ne sont pas bonnes ! Quand vous voulez utiliser des couleurs de PEN et de PAPER différentes dans des fenêtres, examinez bien les couleurs d'abord, et vérifiez qu'elles sont réellement compatibles. Quelquefois la combinaison de deux teintes d'une même couleur peut être très efficace, mais deux couleurs pâles ou deux couleurs sombres ne seront jamais bien satisfaisantes, surtout si vous utilisez un téléviseur couleur. Certaines de ces combinaisons ne sont pas satisfaisantes, même sur un moniteur.

Un meilleur affichage

```
10 CLS
20 INK 2,6,8
30 INK 3,12,19
40 PEN 2
50 PRINT:PRINT "ATTENTION S.V.P.!"
60 PRINT:PRINT:PRINT
70 PEN 3
80 PRINT "ATTENTION : DANGER !"
90 PEN 1
```

Fig. 8.8. Texte clignotant obtenu par l'usage d'encre clignotante !

Pour commencer notre nouvelle exploration des effets spéciaux de texte, le mieux est de partir à nouveau d'INK. Commencez par regarder le programme de la *figure 8.8* qui illustre le texte clignotant. Le clignotement est produit par l'usage d' « encre clignotante » dans les pots 2 et 3, aux lignes 20 et 30. Quand on utilise PEN pour tremper notre plume dans ces pots, le résultat est un texte clignotant, comme vous pouvez le voir à l'exécution des lignes 50 et 80. La ligne 90 revient à PEN 1, l'encrier normal pour du texte, de sorte que le message « Ready » ne clignote pas. Notez que les lettres restent clignotantes tant qu'elles sont sur l'écran. Vous pouvez changer instantanément les couleurs clignotantes, ou passer à une couleur fixe, en modifiant INK.

La *figure 8.9*, par exemple, montre comment un titre peut clignoter pendant quelques secondes, puis revenir à une couleur

fixe. C'est une bonne manière de vous assurer que le clignotement attire bien l'attention de l'utilisateur, mais ne le gêne pas ensuite.

```
10 INK 2,24,6
20 CLS
30 PRINT:PEN 2
40 PRINT TAB(13)"TITRE CLIGNOTANT"
50 FOR N=1 TO 3000:NEXT
60 INK 2,20
70 PEN 1
80 PRINT:PRINT"Maintenant,on peut mettre
 du texte sur":PRINT "l'écran sans etre
distrait!"
```

Fig. 8.9. Passage de l'affichage clignotant à l'affichage fixe en changeant INK.

Un autre trait utile du CPC464 vous permet de créer des effets comme le soulignement, l'accentuation, etc. Ceci, comme beaucoup d'autres effets que nous verrons plus tard, est rendu possible par un des codes « invisibles » du CPC464. Il y a dans cette catégorie les codes 1 à 31, dont nous avons déjà utilisé le code 21 (couper la sortie écran), le code 6 (autoriser la sortie écran) et le code 8 (ramener le curseur en arrière d'une colonne).

Votre manuel présente une liste complète de ces effets au chapitre 9, page 2. Pour souligner du texte, comme c'est illustré à la *figure 8.10,* nous devons changer le mode d'affichage.

```
10 CLS
20 PRINT:PRINT:PRINT
30 PRINT "Ceci est important";
40 S$=STRING$(19,8)
50 PRINT S$;
60 PRINT CHR$(22)+CHR$(1);
70 PRINT"____ ___ _________"
80 PRINT:PRINT
90 PRINT CHR$(22)+CHR$(0)
```

Fig. 8.10. Soulignement, utilisant CHR$(22) pour éviter l'effacement par surimpression. Notez aussi comment STRING$ peut être utilisé avec un code ASCII à la place d'un caractère entre guillemets.

Normalement, quand on affiche, le nouvel affichage remplace complètement l'ancien. En utilisant CHR$(22)+CHR$(1), on peut modifier cela, en surimposant le nouveau texte sur l'ancien. Dans le programme, donc, l'expression de la ligne 30 est affichée, et la chaîne S$ est définie comme un recul de 19 colonnes. Cela placera le curseur au début de l'expression. Ensuite on met en route la surimpression avec CHR$ (22) + CHR$ (1), et on affiche un ensemble de soulignés. Dans cet état de la machine, les soulignés sont ajoutés, ils ne remplacent pas le texte.

Si nous n'annulions pas cet état, cela pourrait donner des effets bizarres, au cas où nous voudrions taper un nouveau texte, ou éditer quelque chose aussi faut-il arrêter la surimpression en utilisant PRINT CHR$(22)+CHRS$(0). Beaucoup d'instructions de ce type nécessitent deux termes en CHR$ comme celui-ci (ou plus), et cet exemple montre la manière de les joindre pour rendre l'instruction complète. Beaucoup de ces instructions effectuent les mêmes opérations que des instructions BASIC, et leur utilité réside dans le fait qu'elles permettent d'effectuer les opérations en utilisant des paramètres numériques. Si vous avez utilisé le micro BBC, vous pouvez reconnaître cette procédure : elle est l'équivalent des instructions « VDU » du BBC.

De choses et d'autres

Il y a beaucoup d'instructions sur cette machine qui ne tombent pas clairement dans une catégorie définie. Prenez par exemple LOWER$ et UPPER$ *(figure 8.11).* Chacune de ces instructions doit être suivie par le nom d'une variable chaîne, entre parenthèses.

```
10 CLS:PRINT:PRINT TAB(10)"MAJUSCULES-MI
NUSCULES"
20 PRINT:PRINT
30 FOR N=1 TO 5
40 INPUT "NOM- ";A$
50 PRINT UPPER$(A$),LOWER$(A$)
60 NEXT
```

Fig. 8.11. Conversion majuscules/minuscules avec UPPER$ et LOWER$.

Quand vous utilisez UPPER$, toutes les lettres de la variable chaîne sont converties en majuscules. Quand vous utilisez LOWER$, toutes les lettres de la variable chaîne sont converties en minuscules. C'est utile si vous avez des entrées mélangées, en minuscules et en majuscules, et que vous vouliez les convertir pour les unifier.

Pourquoi faire cela ? Eh bien, une bonne raison pourrait être l'ordre alphabétique. Comme nous l'avons vu, le classement met les mots en ordre des codes ASCII. Les codes ASCII pour les minuscules sont tous plus grands que ceux des majuscules, cependant. Donc l'ordinateur, sans intervention, rangerait TOUT,tout,BON,binaire dans l'ordre BON,TOUT, binaire, tout. En convertissant tous les caractères, on réalisera un classement plus logique.

Incidemment, tant que nous en sommes au classement des chaînes, il y a une instruction utile, FRE. Si vous tapez en commande directe, ou placez dans un programme PRINT FRE(0), l'ordinateur vous dira combien il vous reste de mémoire. J'aurais bien aimé que mes autres ordinateurs m'en offrent autant ! Mais si vous utilisez FRE(""), l'ordinateur va réorganiser le stockage des chaînes. C'est important parce que durant un tri de chaînes la mémoire devient encombrée d'espaces inutilisés, là où des chaînes ont été créées pour des échanges. FRE("") libère cet espace, et son usage durant un long programme de tri peut rendre le programme plus rapide et plus efficace.

Vous voulez afficher de l'espace ? Il y a une instruction chaîne commode, SPACE$, qui le fera pour vous. Par exemple S$=SPACE$(40) vous donnera une ligne de 40 espaces quand vous afficherez S$. C'est plus facile que de créer une boucle pour remplir une chaîne de blancs.

Une instruction plus ésotérique est WRITE. Si vous utilisez WRITE " PAPIER " à la place de PRINT " PAPIER ", les *guillemets sont affichés en même temps que le mot !* La même chose arrive si vous utilisez A$=" PAPIER " puis WRITE A$. C'est utile si vous voulez voir apparaître des guillemets dans une expression, car c'est normalement impossible avec une instruction PRINT.

Un dernier point. Les touches du CPC464 se répètent quand vous les maintenez appuyées. Pour beaucoup d'usages, la

vitesse par défaut est idéale, mais il se peut que vous vouliez la modifier. Par exemple, vous pouvez trouver que le rythme de répétition est trop lent dans certains jeux, ou trop rapide pour certains programmes de traitement de données. Dans quelques cas, vous pouvez souhaiter que le rythme soit si lent, qu'il soit pratiquement impossible de répéter automatiquement une touche. Si, par exemple, vous avez deux instructions INKEY$ l'une après l'autre, tenir la touche enfoncée trop longtemps au premier INKEY$ risque de faire que le même choix sera pris pour les deux cas, et de vous entraîner dans un mauvais choix sur un menu, par exemple.

On peut régler le rythme de répétition en utilisant SPEED KEY suivi de deux nombres. Le premier mesure le temps que vous devez maintenir la touche enfoncée avant que la répétition ne commence. Le deuxième décide de l'intervalle entre les répétitions. L'intervalle acceptable va de 0 à 255 ; si vous choisissez des nombres trop faibles, vous risquez d'avoir des effets bizarres, comme des doubles lettres apparaissant alors que vous n'avez tapé une touche qu'une seule fois.

Essayez de taper, d'abord avec SPEED KEY 2,2, puis avec SPEED KEY 255,255 (si toutefois vous arrivez à taper cette seconde commande !). Le premier exemple rend presque impossible de taper normalement, le deuxième interdit pratiquement la répétition pour tout usage efficace.

9

Premiers pas en graphique

On attend de tout ordinateur moderne la capacité de produire des affichages éblouissants en couleur, et d'autres effets spéciaux. Le CPC464 ne fait pas exception, et nous allons commencer à voir dans ce chapitre quelques-uns des effets graphiques possibles. Pour commencer, il nous faut connaître quelques termes usuels.

Le premier, *graphique,* signifie les images qu'on peut dessiner sur l'écran, et tous les ordinateurs modernes ont des instructions qui permettent de dessiner des motifs. Dans ce domaine, vous verrez utiliser les termes de « basse » et « haute résolution ». Le terme de *résolution* n'est pas très facile à expliquer. Imaginez que vous dessiniez des images sur une feuille de papier d'environ 24 cm de large par 17 cm de haut — c'est en gros la taille d'un écran TV décrit comme un « 30 cm » (on donne la taille de la diagonale).

Maintenant si on vous demande de créer ces images en utilisant des rectangles colorés de papier, vous faites votre dessin d'une manière assez semblable à celle de l'ordinateur. Supposez qu'on ne vous autorise à utiliser que 936 rectangles de papier, d'une taille propre à remplir l'écran à eux tous. Vous ne pourriez pas dessiner d'images très finement détaillées, avec un si petit nombre de morceaux, d'une taille assez grande. C'est ce qu'on entend par *basse résolution.* Un certain nombre d'ordinateurs n'offrent guère mieux. D'un autre côté, si on vous donnait des morceaux si petits qu'il en faudrait 32000 pour remplir un écran, vous pourriez produire des images bien plus détaillées. C'est ce qu'on entend par *haute résolution.*

Le CPC464 offre des instructions graphiques de haute résolution, et le chiffre de 32000 éléments que j'ai employé correspond à la taille des blocs que le CPC464 peut employer dans son *mode de plus basse résolution,* le mode 0. Dans son mode de plus haute résolution, le CPC464 peut travailler avec 128 000 éléments sur l'écran. Pour créer ces images graphiques, un ordinateur doit utiliser de la mémoire. Beaucoup d'ordinateurs utilisent à cet effet la même mémoire que celle accessible aux programmes des utilisateurs. Aussi beaucoup d'ordinateurs dont on dit qu'ils ont 64K ou 32K de mémoire peuvent en fait être réduits à bien moins en graphique, quelquefois de l'ordre de 6K ! La raison en est que beaucoup de mémoire sert à assurer le graphique et les autres systèmes d'exploitation. Le CPC464 ne vous vole pas de mémoire programme pour les besoins du graphique. A la place, il a un espace de mémoire réservé *(dédié)* au graphique et les 43533 octets (42.5K) de mémoire pour programme ne sont pas accaparés par des usages graphiques.

Jusqu'à maintenant, nous avons travaillé avec ce qu'on appelle « l'écran texte », qui constitue l'état normal de l'écran. Cet écran texte ne peut être utilisé que pour du texte, à savoir des caractères mis en place par des instructions PRINT. Comme nous le verrons, il y a d'autres instructions utilisables avec les « écrans graphiques », mais nous y viendrons plus tard.

Clavier graphique

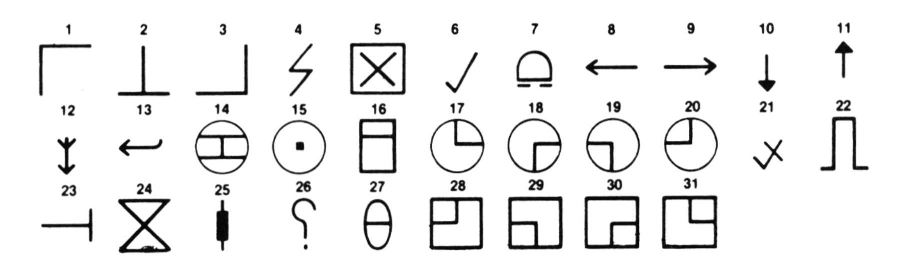


Fig. 9.1. Les symboles graphiques qu'on peut obtenir par une touche, avec leur code ASCII.

Certains symboles graphiques, illustrés à la *figure 9.1*, peuvent être obtenus en enfonçant simplement des touches du clavier. La différence est que vous devez enfoncer la touche CTRL en même temps. Ces caractères graphiques *ne peuvent* toutefois *pas* être affichés comme vous affichez des mots, en utilisant l'instruction PRINT suivie de guillemets, puis en frappant les caractères graphiques, et en refermant les guillemets.

Si vous voulez utiliser ces caractères pour faire un soulignement recherché, ou comme source de motifs d'identification de choix dans un menu, vous devez utiliser une méthode différente pour les introduire dans vos programmes. La solution consiste à utiliser les nombres codes correspondant aux caractères voulus.

Les codes des caractères graphiques

La solution, qui nous permet bien plus de champ pour l'illustration, consiste à utiliser les codes ASCII des caractères. Maintenant il y a deux ensembles de codes ASCII qui produisent des motifs graphiques. Un ensemble utilise les codes ASCII de 128 à 255. Les motifs obtenus sont montrés dans l'*appendice III* du manuel du CPC464, mais le programme de la *figure 9.2* vous les rappellera. On trouve plusieurs lettres de l'alphabet grec dans cet ensemble, qui pourront être utiles à des fins scientifiques ou techniques. Le programme est arrangé pour afficher les formes séparées l'une de l'autre, et il serait intéressant que vous examiniez comment cela a été obtenu.

```
10 CLS:N= 127
20 FOR J=1 TO 128
30 PRINT CHR$(N+J);" ";
40 IF J/20=INT(J/20) THEN PRINT
50 NEXT
```

Fig. 9.2. Un programme qui vous montrera les caractères graphiques correspondant aux codes 128 à 255.

Les autres symboles graphiques sont obtenus avec des codes ASCII de 1 à 31. Le hic est ici que ces codes sont déjà utilisés pour d'autres effets, et vous devez savoir comment passer d'un usage à l'autre. Le secret est CHR$(1). Si vous faites précéder n'importe lequel de ces codes de 1 à 31 par CHR$(1), le motif sera affiché *sans* les autres effets que vous pourriez attendre. Par exemple, vous savez que PRINT CHR$ (8) fait reculer le curseur d'une colonne. Mais PRINT CHR$(1) + CHR$(8) affichera simplement un motif graphique. Le programme de la *figure 9.3* révèle ces formes, dont plusieurs sont utiles aussi bien dans des programmes d'affaires ou d'éducation que dans des jeux.

```
10 CLS
20 FOR N=1 TO 31
30 PRINT N;" ";CHR$(1)+CHR$(N);"  ";
40 IF N/5=INT(N/5) THEN PRINT:PRINT
50 NEXT
```

Fig. 9.3. Un programme qui révèle les caractères graphiques des codes 1 à 31. Notez qu'un simple PRINT CHR$(X), avec X entre 1 et 31 ne produit pas ces symboles.

Vous pouvez faire du travail de décoration avec ces motifs en CHR$ si vous utilisez une des grilles du manuel pour établir votre plan. Les grilles en MODE 1 (*appendice VI,* page 2 du manuel) donnent 40 cases en largeur sur 25 en hauteur, car c'est le nombre de positions de caractères disponible en Mode 1. Si vous utilisez l'écran du Mode 0, les caractères seront plus larges, parce qu'il n'y a que 20 caractères par ligne dans ce mode. Si vous utilisez le Mode 2, vous pourrez tasser jusqu'à 80 caractères sur une ligne.

Notez toutefois que les caractères ont toujours la même hauteur, seule leur largeur change. Les motifs apparaissent donc assez différents dans les trois modes. Le Mode 0 donne des caractères en « Cinémascope », le Mode 2 les amaigrit ! Chaque case de la grille représente la position d'un caractère, et si vous dessinez ce que vous voulez sur une feuille de calque placée au-dessus de la grille vous pourrez imaginer à quoi ressemblera la forme sur l'écran.

Vous avez trois méthodes pour programmer cela. L'une

consiste à afficher chaque ligne de motifs séparément. Une autre utilise une boucle pour l'affichage, avec les codes numériques stockés sur des lignes de DATA. La troisième méthode est de placer tous les caractères dans une chaîne, tout comme vous pouvez faire une seule chaîne de plusieurs mots.

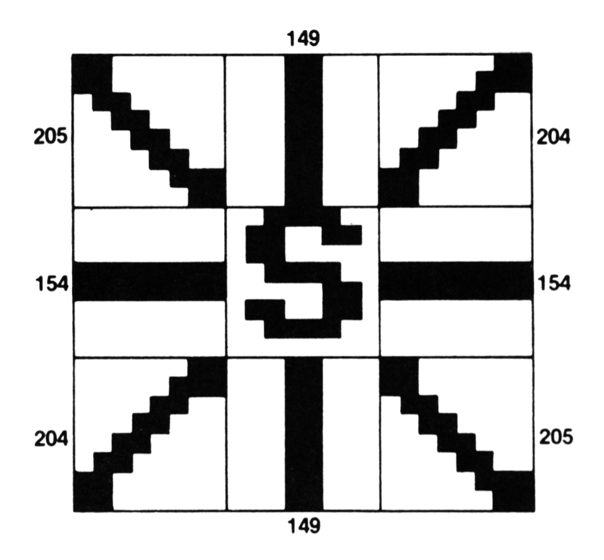


Fig. 9.4. Utilisation des symboles graphiques pour dessiner un logo.

Oui, je sais, un exemple serait utile. La *figure 9.4* montre la conception d'un « logo », la marque d'identification d'une firme, par exemple. Ce logo consiste en la lettre S d'où rayonnent des lignes. La lettre S ne fait pas de problème, c'est CHR$(83), mais les autres caractères sont pris dans l'ensemble donné par le manuel. Le plan a été fait en traçant sur la grille les motifs du manuel. Maintenant vous pouvez simplement écrire un programme qui affiche chaque valeur de CHR$ au bon emplacement, comme le montre la *figure 9.5.* Cela marche, mais c'est de la programmation maladroite.

```
5 PAPER 3:PEN 2
10 CLS:PRINT:PRINT
20 PRINT TAB(5) CHR$(205)+CHR$(149)+CHR$
(204)
```

```
30 PRINT TAB(5) CHR$(154)+CHR$(83)+CHR$(
154)
40 PRINT TAB(5) CHR$(204)+CHR$(149)+CHR$
(205)
50 PRINT:PRINT
```

Fig. 9.5. Un programme simple pour afficher le logo de la *figure 9.4.*

Regardez maintenant la *figure 9.6.* Peut-être qu'elle ne vous paraît pas plus nette, il faut plus de lignes, par exemple, mais elle est quand même bien meilleure : il y a une seule instruction PRINT CHR$ (D%), à la place de trois. On utilise deux boucles, une par ligne de caractères et une pour chaque colonne. Toutes les variables numériques sont des variables entières (utilisant le symbole %), si bien que le programme peut tourner plus vite. De plus, la position du logo est décidée par l'utilisation de LOCATE, dans la boucle. L'avantage de cette méthode est que vous voyez clairement les données, et qu'ainsi elles sont faciles à modifier, sans toucher au programme.

```
10 PAPER 3:PEN 2:CLS
20 FOR J%=1 TO 3:LOCATE 18,J%+10
30 FOR N%=1 TO 3
40 READ D%:PRINT CHR$(D%);
50 NEXT:PRINT:NEXT
60 WINDOW#2,1,40,24,25
70 WINDOW SWAP 0,2
100 DATA 205,149,204
110 DATA 154,83,154
120 DATA 204,149,205
```

Fig. 9.6. Utilisation de boucles pour lire des caractères graphiques.

Notez que j'ai utilisé une fenêtre, échangée avec le canal #0, pour faire apparaître le message « Ready » et le curseur au bas de l'écran, et non juste sous le logo. Vous devrez faire un autre échange WINDOW SWAP 0,2 après l'exécution de ce programme pour récupérer la liste du programme en plein écran.

```
10 PAPER 2:PEN 3:CLS
20 GR$="":B$=STRING$(3,8)+CHR$(10)
30 FOR J%=1 TO 3:FOR N%=1 TO 3
40 READ D%:GR$=GR$+CHR$(D%):NEXT
50 gr$=gr$+B$:NEXT
60 LOCATE 18,12:PRINT GR$
70 DATA 205,149,204,154,83,154,204,149,2
05
```

Fig. 9.7. Réunion de caractères graphiques en une chaîne affichable en une seule instruction.

La *figure* 9.7 illustre une méthode encore meilleure. Elle commence par définir une chaîne appelée B$. Celle-ci consiste en trois caractères de code 8, le retour arrière, suivis du code 10. Si vous cherchez cela dans le manuel, vous verrez que 10 est le caractère « descente d'une ligne ». L'effet de l'affichage de B$ est donc de mettre le curseur trois colonnes en arrière (à gauche) et une ligne en dessous. A la ligne 20, aussi, la chaîne GR$ est assimilée à une chaîne vide. Ensuite débutent deux boucles, l'une pour les lignes, l'autre pour les colonnes. Après la lecture d'une ligne de DATA, et l'ajout des données à GR$ à la ligne 40, on concatène B$. Cela fera descendre le curseur d'une ligne et le fera reculer de trois colonnes à l'affichage. L'effet global est donc d'afficher trois caractères, de mettre le curseur à la bonne position à la ligne suivante pour afficher les caractères suivants. On ajoute chaque ligne à la chaîne, et enfin on affiche le tout à la ligne 60.

Le grand avantage de cette méthode illustrée à la *figure* 9.7 est que le motif peut être affiché n'importe où sur l'écran sans autre modification du programme. N'importe quelle instruction PRINT GR$ affichera le motif complet, quel que soit l'emplacement du curseur. Vous devez donc évidemment veiller à ne pas placer le curseur trop à droite ou trop bas sur l'écran.

Armés de cette technique pour produire des motifs, voyons maintenant comment on peut les faire apparaître avec du mouvement, de l'animation. C'est illustré à la *figure* 9.8. La méthode est d'afficher le motif à une position déterminée par LOCATE. Le motif est laissé à l'écran un court instant, puis effacé. Après un autre instant, la position de LOCATE est

```
10 PAPER 2:PEN 3:CLS
20 GR$="":B$=STRING$(3,8)+CHR$(10)
30 FOR J%=1 TO 3:FOR N%=1 TO 3
40 READ D%:GR$=GR$+CHR$(D%):NEXT
50 GR$=GR$+B$:NEXT
60 A$=STRING$(3,32):SP$="":SP$=A$+B$+A$+
B$+A$
70 FOR N=1 TO 18
80 LOCATE N,12
90 PRINT GR$:GOSUB 1000
100 LOCATE N,12
110 PRINT SP$:GOSUB 1000
120 NEXT:LOCATE N,12:PRINT GR$
130 DATA 205,149,204,154,83,154,204,149,
205
140 END
1000 FOR K=1 TO 20:NEXT:RETURN
```

Fig. 9.8. Animation simple sur l'écran texte.

décalée vers l'emplacement suivant, et le processus est répété. Si le délai est court, comme dans cet exemple, et le mouvement petit, ce qui *n'est pas le cas* dans cet exemple, l'illusion de mouvement continu est bonne. Mais l'animation n'est pas parfaitement adaptée à l'écran texte, sur lequel les positions des caractères sont trop éloignées.

Créez vos propres caractères !

Le CPC464 offre une manière intéressante de créer du graphique sur l'écran texte ordinaire, en utilisant simplement l'instruction PRINT pour placer les motifs. Ces motifs peuvent être classés comme de « basse résolution » dans le sens qu'ils utilisent le nombre limité de positions de PRINT sur l'écran, mais ils offrent beaucoup plus de latitude pour des effets visuels que les caractères prédéfinis.

Comme le titre de section le suggère, on peut créer ses propres caractères. Il y a deux points à voir : comment préparer les motifs, et comment les placer sur l'écran. Procédons méthodiquement.

Nous commencerons logiquement par la préparation. La

taille du motif dont nous parlons est celle d'un caractère à l'écran, la taille du bloc du curseur. En fait celui-ci, et tous les autres caractères du CPC464 sont constitués de 64 points arrangés sur une grille de 8 sur 8. La *figure 9.9* vous montre la forme de cette grille — vous pouvez la redessiner pour vous-même sur du papier millimétré si vous en voulez des copies.

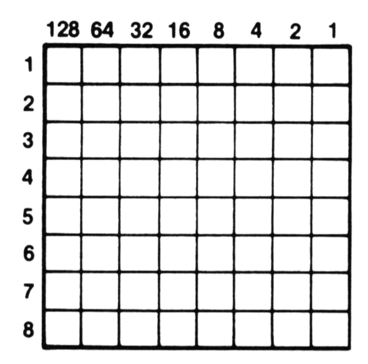


Fig. 9.9. La grille d'un caractère, pour dessiner vos propres symboles.

Le point important est que les petits carreaux de la grille représentent des points de l'écran qui peuvent être de la couleur d'INK ou de PAPER suivant la valeur de codes numériques que nous fournissons à l'ordinateur.

Quand un caractère est conçu sur cette grille 8 × 8, on utilise normalement 7 point par 7, en laissant la colonne de droite et la rangée du bas inutilisées. C'est pour avoir un espace entre deux caractères et entre deux lignes de texte sur l'écran. Si vous concevez vos propres graphiques, cependant, vous pourriez désirer leur faire remplir complètement le bloc 8 × 8, et ainsi joindre plusieurs motifs à l'affichage.

Le manuel du CPC464 vous montre certes brièvement comment concevoir ces motifs, mais si vous ne l'avez jamais fait auparavant, vous pouvez rester intrigué. La clé du procédé est dans les nombres imprimés en haut de chaque colonne de carreaux à la *figure 9.9.* Chaque nombre est un code pour les carreaux de la colonne qu'il domine. Utilisez ce nombre, et le carreau sera de la couleur d'INK. Utilisez 0 à la place, et le carreau sera de la couleur de PAPER, c'est-à-dire invisible.

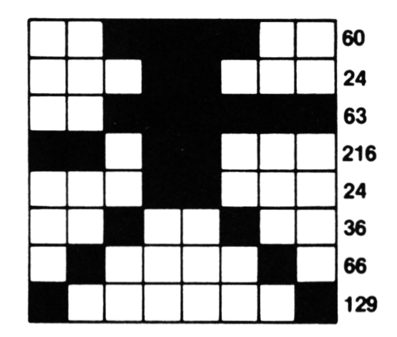


Fig. 9.10. Comment utiliser la grille : plan d'un « envahisseur de l'espace ».

Un exemple aidera à rendre les choses plus claires, à la *figure 9.10.* J'ai utilisé le motif simple d'un « envahisseur de l'espace » pour illustrer le principe.

La première ligne de carreaux a juste quatre carreaux de noircis. Je travaille d'ordinaire sur du papier calque fixé par un trombone au-dessus de la grille, mais dans cet exemple j'ai montré ce à quoi cela ressemblerait sur le papier millimétré même. Les carreaux noircis de la ligne du haut sont ceux que nous voulons voir apparaître en couleur d'INK, et ils sont sous les nombres 32,16,8 et 4. Il n'y a rien d'autre de noirci sur cette ligne, aussi il nous suffit d'additionner les nombres 32,16,8 et 4, ce qui fait 60, et de noter le résultat sur le côté.

On procède de manière similaire pour la deuxième ligne : les carreaux des positions 16 et 8 sont noircis et ainsi le nombre que nous utilisons est la somme, 24. On continue ainsi, jusqu'à la fin des huit lignes.

Il y a quelques points à noter. L'un est que si aucun des carreaux n'est noirci, le code numérique est zéro. L'autre est que vous pouvez éviter beaucoup de calculs en vous souvenant qu'une rangée complètement noircie fait un total de 255. C'est la taille maximum du nombre que vous pouvez utiliser comme code numérique pour un caractère de texte. Vous devez toujours avoir à la fin huit nombres, quel que soit le motif que vous projetiez de dessiner.

Le point suivant est de demander à l'ordinateur de produire le motif. Nous devons stocker les codes numériques dans la mémoire du CPC464, avec un code ASCII que nous

utiliserons pour obtenir la forme sur l'écran. On fait usage d'une nouvelle instruction, SYMBOL.

SYMBOL doit être suivi de *neuf* nombres. Le premier nombre est le code ASCII que nous voulons utiliser. Ainsi, en appuyant sur une touche ou en utilisant ce code ASCII, on obtiendra notre motif. Les huit nombres suivants sont les nombres codes obtenus sur la grille. Tous les nombres sont séparés par des virgules, il doit y avoir un espace entre le L de SYMBOL et le premier nombre, et les nombres du motif sont lus de haut en bas. L'effet de SYMBOL est de placer le code numérique en mémoire.

Tout cela peut paraître simple, mais quels codes ASCII pouvons-nous utiliser ? La réponse est que vous disposez des codes ASCII 240 à 255, un total de 16, à cet usage. Douze de ces codes sont normalement fournis par les touches de curseur en haut à droite du clavier (il y a douze codes parce que les quatre touches peuvent être associées aux touches SHIFT et CTRL).

```
10 CLS
20 SYMBOL 244,60,24,63,216,24,36,66,129
30 PRINT:PRINT TAB(12) CHR$(244)
```

Fig. 9.11. Apparition de l'envahisseur.

Utiliser les codes pour votre propre usage, cependant, *n'affecte pas l'usage normal de ces touches.* Vous n'obtiendrez pas d'envahisseur en appuyant sur une touche du curseur ! Supposons que nous voulions faire apparaître notre envahisseur quand on utilise CHR$(244). La *figure 9.11* montre ce qu'il faut, et ce n'est pas grand-chose ! Le code ASCII est 244, et ce nombre est suivi des huit que nous avons définis sur la grille du motif. On peut ensuite produire la forme n'importe où en utilisant PRINT CHR$(244).

Maintenant vous n'êtes pas limités à la création de caractères pour l'usage avec les codes ASCII 240 à 255. Vous pouvez utiliser n'importe quel nombre de départ, à partir de 32. Cela signifie que vous pouvez redéfinir n'importe quel code de touche, et ainsi faire des caractères qui apparaîtront quand

vous appuyez sur des touches ordinaires ! La *figure 9.12* montre ce qui est nécessaire pour cela.

```
10 CLS:SYMBOL AFTER 90
20 SYMBOL 92,60,24,63,216,24,36,66,129
30 PRINT:PRINT TAB(12)"\"
```

Fig. 9.12. Redéfinition du code d'une touche, pour qu'elle donne le symbole « envahisseur ».

A la ligne 10, nous avons maintenant ajouté SYMBOL AFTER 90. Cette instruction signifie que nous pouvons faire nos propres caractères (les « redéfinir ») pour tout caractère de code ASCII égal ou supérieur à 90. Nous avons choisi 92, qui est le code ASCII de la barre oblique inversée appelée quelquefois « anti-slash »,\ , la touche à côté du SHIFT de droite. Quand vous exécuterez ce programme, vous verrez l'envahisseur apparaître en réponse à l'instruction PRINT TAB(12) ; " \". Vous pouvez bien sûr aussi utiliser PRINT CHR$(92) si vous voulez. Mais le plus important est que vous verrez l'envahisseur apparaître à chaque appui sur la touche

Pour revenir à l'état normal du clavier, sauvez votre programme, et appuyez sur SHIFT, CTRL et ESC pour restaurer l'état de la machine.

```
10 CLS:SYMBOL AFTER 32
15 FOR N=32 TO 127
20 SYMBOL N,60,24,63,216,24,36,66,129
25 NEXT
```

Fig. 9.13. Redéfinition de toutes les touches — cela rend la suite de la programmation difficile ! Appuyez sur CTRL-SHIFT-ESC pour revenir à un état normal, mais sauvez votre programme avant.

Il y a encore un point important. Aucun des autres codes supérieurs à 90 n'est affecté par ce changement, le seul caractère redéfini est celui de code 92. Évidemment, si vous le voulez, il n'y a aucune raison qui vous empêche de redéfinir toutes les touches, mais c'est une rude tâche. Rien que pour

le plaisir, jetez un coup d'œil à la *figure 9.13* qui redéfinit toutes les touches comme notre envahisseur. Après l'exécution de ce programme, tout ce qui se trouve sur l'écran sauf le curseur est un envahisseur ! Même les messages comme « Syntax error » et « Ready » apparaissent comme une chaîne d'envahisseurs ! C'est une bonne manière de vous assurer que personne ne vienne toucher à votre ordinateur en votre absence ! Vous devrez utiliser SHIFT, CTRL et ESC pour retrouver des services normaux ; n'oubliez pas que cela effacera votre programme de la mémoire.

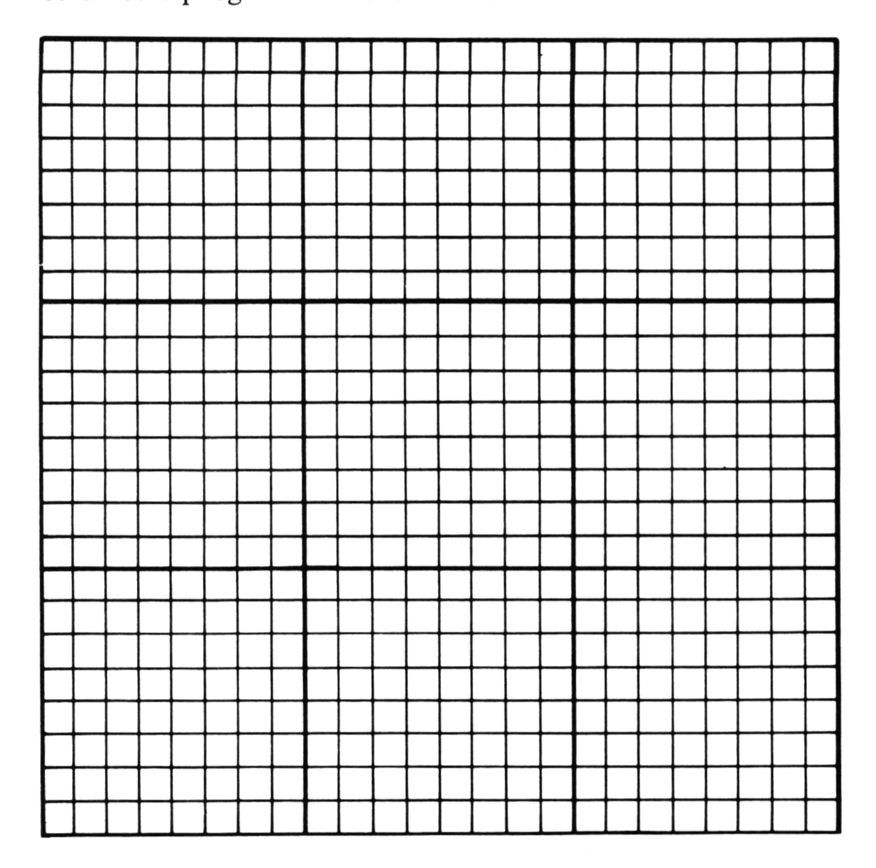


Fig. 9.14. Une grille multi-caractères, pour faire des symboles à partir de blocs de caractères.

Vous n'êtes pas restreints à la création d'un seul caractère de cette manière, vous pouvez aussi créer des caractères qui se

combinent pour former un motif plus grand ! La *figure 9.14* montre une grille multi-caractères qui vous permet de préparer des motifs faits de 9 éléments. Supposez par exemple que nous fassions de nouveaux caractères pour les codes 240 à 242. Nous pourrions ensuite placer ces trois caractères dans une chaîne, GR$, et l'instruction PRINT GR$ afficherait le motif complet !

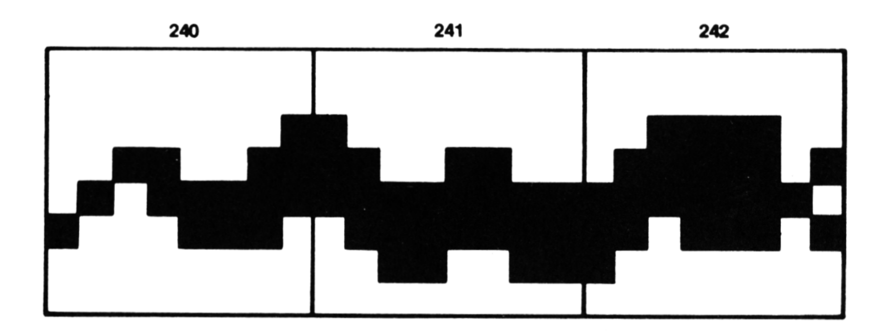


Fig. 9.15. Une forme de serpent dessinée à partir de trois caractères.

```
10 CLS
20 SYMBOL 240,0,0,1,51,95,142,0,0
30 SYMBOL 241,0,0,128,204,255,127,51,0
40 SYMBOL 242,0,0,60,125,254,221,128,0
50 GR$=GR$+CHR$(240)+CHR$(241)+CHR$(242)
60 X=2:Y=12
70 EVERY 10 GOSUB 1000
80 FOR N=1 TO 25:LOCATE 5,N:PRINT"SSSS S
ERPENT"
90 FOR N=1 TO 1000:NEXT:NEXT
100 GOTO 100
110 END
1000 LOCATE X-1,Y:PRINT " ":LOCATE X,Y
1010 PRINT GR$;
1020 X=X+1
1030 IF X>38 THEN LOCATE X-1,Y:PRINT CHR
$(18);:X=2
1040 RETURN
```

Fig. 9.16. Animation du serpent, utilisant EVERY.

La *figure 9.15* montre un motif dessiné sur les trois grilles accolées. La *figure 9.16* montre ensuite la réalisation, et comme vous savez maintenant à peu près tout à ce sujet, j'évite les détails. J'ai utilisé les codes 240 à 242 inclus, et alloué les motifs à chacun. Comme la grille 8 × 8 a été utilisée en entier, les parties du « serpent » seront accolées à l'affichage. On affiche le résultat en combinant les trois caractères dans une chaîne. C'est simple, ardu, et fascinant à regarder !

Les interruptions périodiques automatiques

Le programme précédent illustre aussi une instruction entièrement nouvelle, EVERY. Cette instruction est très importante pour l'animation, et nous allons l'examiner en détail, avec d'autres instructions apparentées.

EVERY vous permet de faire usage d'une des horloges du CPC464, qui sont quatre en tout. L'idée est que vous pouvez spécifier que quelque chose se produise périodiquement. La période dépend du nombre qui suit EVERY. Pour les versions à 50 hertz du CPC464, un nombre de 50 spécifie une action par seconde. A cet intervalle par exemple, vous pouvez spécifier l'exécution d'un sous-programme.

Le point important est que le sous-programme sera exécuté à chaque intervalle, *même quand l'ordinateur fait autre chose !* Vous verrez dans l'exemple qu'une boucle est exécutée — elle affiche " SSSS SERPENT " — pendant que le serpent traverse l'écran. Quand la boucle est achevée, il y a encore une boucle infinie, exécutée à la ligne 100. Cela maintient le programme, et par conséquent l'horloge, en marche. L'horloge travaille indépendamment des autres parties du programme, et peut interrompre l'action du programme pour effectuer le sous-programme selon la période spécifiée. Dans cet exemple l'intervalle est de 10, un cinquième de seconde. Un motif qui peut être animé indépendamment des autres effets du programme de cette manière est appelé un *lutin*, (ou *sprite* selon le terme anglais).

Beaucoup de machines vous permettent de nos jours d'utiliser des lutins, mais peu vous donnent dessus un contrôle aussi simple et efficace que le CPC464.

Comme il y a quatre horloges, vous pouvez avoir au moins quatre ensembles d'objets bougeant indépendamment — chaque horloge peut contrôler autant de lutins que vous le désirez, évidemment. Pour spécifier l'horloge que vous voulez utiliser, le numéro d'horloge est placé après le nombre fixant la période, séparé par une virgule. On utilise les nombres de 0 à 3. Vous pouvez, par exemple, utiliser des instructions comme EVERY 10,0 GOSUB 1000 ou EVERY 20,3 GOSUB 2000.

```
10 CLS
20 FOR Y=1 TO 25
30 PRINT TAB(RND(1)*35);POS(#0);",";VPOS
(#0)
40 FOR N=1 TO 1000:NEXT
50 NEXT
```

Fig. 9.17. Utilisation de POS et VPOS pour trouver et afficher la position du curseur.

Pendant que nous en sommes au déplacement d'objets sur l'écran « texte », le CPC464 vous offre un moyen de trouver où se trouve le curseur. Cela requiert l'utilisation des mots POS et VPOS, qui *doivent* être suivis d'un numéro de fenêtre. Pour l'écran entier, on ne peut omettre le #0. La *figure 9.17* montre une illustration simple du fonctionnement de POS et VPOS. POS donne la position horizontale du curseur, et VPOS la verticale. Les nombres qui sont utilisés sont analogues à ceux de l'instruction LOCATE.

POS et VPOS sont particulièrement utiles quand vous travaillez avec des fenêtres, parce qu'il n'est pas toujours facile de se représenter où se trouve le curseur. Vous pouvez utiliser les nombres fournis par POS et VPOS dans tous les tests que vous voulez.

10

Guide du graphisme évolué

Les possibilités graphiques du CPC464 sont bien plus étendues que celles de beaucoup de machines concurrentes. Dans ce chapitre, nous allons examiner les instructions qui s'appliquent spécifiquement au graphisme, en ce sens qu'elles permettent de dessiner sans passer par PRINT ou CHR$.

Le premier point à noter est que la définition de ces instructions graphiques du CPC464 est très haute. Si vous regardez de près l'écran d'un téléviseur couleur, vous verrez qu'il est divisé en une série de lignes ou de points. Ce sont ces éléments qui brillent et donnent de la lumière, et on ne peut rien afficher de plus petit sur un écran qu'un point d'écran, ou de plus rapproché que la distance qui sépare deux lignes. En fait, à moins d'utiliser un moniteur, on ne peut même pas approcher la taille d'un point, ou l'étroitesse d'une ligne.

Les « points » que peut utiliser un ordinateur sont appelés *pixels* (de l'anglais *pic*ture *el*ements, éléments d'image). Chaque pixel qu'un ordinateur peut contrôler correspond à plusieurs points de l'écran TV. Le CPC464 vous offre trois résolutions au choix, qui correspondent aux trois modes. En mode 0, la résolution graphique est de 160 pixels horizontalement, par 200 verticalement. Dans les autres modes, le nombre de pixels verticalement reste constant, 200. Le mode 1 autorise 320 pixels horizontalement, et le mode 2 enfin 640. La plus haute résolution n'est pas très bien représentée sur un récepteur TV, et vous devez utiliser un moniteur couleur si vous voulez voir de bons résultats en mode 2. C'est une très haute définition, quelle que soit la référence, et le prix de cette haute résolution est la limitation à deux couleurs.

La possibilité de « peindre » chacun des 64000 points (Mode 1) ne serait guère utile si le seul moyen de faire des dessins était de spécifier la position et la couleur de chaque pixel individuellement. Il *existe* des ordinateurs qui n'offrent pas d'autre moyen de créer des images à haute résolution, mais le CPC464, comme vous pouvez vous y attendre, fait plutôt mieux ! Il y a en fait une excellente variété d'instructions graphiques qui vous permettent de dessiner d'une manière plus raisonnable. Toutes ces instructions font usage des coordonnées en X et Y que nous avons déjà rencontrées avec LOCATE. La différence est désormais que les nombres utilisables sont différents.

Pour tous les modes graphiques, l'intervalle acceptable pour les valeurs de X va de 0 à 399. C'est très commode, car cela veut dire que si vous avez développé un programme pour un mode donné, et que vous vouliez l'exécuter dans un autre mode, vous n'avez pas à relire le programme pour changer tous les nombres. La *figure 10.1* vous montre comment tracer une grille graphique sur du papier millimétré.

---

1. Prenez une feuille de papier millimétré de format A4.
2. Marquez les graduations de 25 en 25 jusqu'à 600 sur le grand côté et jusqu'à 400 sur le petit.
3. Utilisez du papier calque par-dessus cette grille pour tracer vos contours et lire les coordonnées.

Fig. 10.1. Comment construire vos propres grilles graphiques sur du papier millimétré.

Il y a d'importantes différences entre les techniques du texte et du graphique. Quand vous travaillez en graphique, vous avez directement affaire aux pixels, si bien que les distances représentées par les unités de X et Y sont bien plus petites que pour le texte. De plus l'origine des coordonnées graphiques est différente. L' « origine » est l'endroit de l'écran désigné par X=0 et Y=0.

Sur l'écran texte, vous ne pouvez pas utiliser zéro, mais LOCATE 1,1 réfère au coin supérieur gauche de l'écran. En graphique, le point X=0, Y=0 (usuellement désigné comme 0,0) se trouve au coin *inférieur* gauche de l'écran. Les distances en Y sont mesurées *vers le haut*, et celles en X vers la

droite. C'est l'emplacement habituel pour la plupart des graphiques. Si vous avez l'habitude de tracer des graphiques, vous vous habituerez facilement aux instructions graphiques. Comme nous en sommes au sujet des graphiques et autres courbes, nous commencerons par une instruction de tracé de graphique, PLOT.

PLOT : point par point

Tracer des graphiques est une fonction graphique essentielle pour des ordinateurs qui peuvent être sérieusement utilisés pour des usages industriels, commerciaux ou éducatifs. Un graphique est une ensemble de points qui peuvent être joints, et qui apportent des informations. Le CPC464 vous tracera des graphiques point par point, en utilisant les instructions PLOT ou PLOTR. La couleur du point marqué dépend de la couleur d'INK en usage au moment de l'exécution de l'instruction.

La différence entre PLOT et PLOTR est que PLOT utilise des coordonnées absolues, alors que PLOTR utilise des coordonnées *relatives à la position du curseur.* Par exemple, si vous utilisez PLOTR 0,0, le point affiché se trouvera là où se trouve le curseur. Le curseur, pour toutes les instructions graphiques, n'est pas le bloc usuel que vous voyez quand vous utilisez du texte. Le curseur graphique est invisible, si bien que vous ne pouvez le repérer à moins d'utiliser des instructions qui laissent des pixels d'une couleur différente du fond sur l'écran. Le curseur ordinaire du texte réapparaît dès qu'un programme graphique s'achève.

La *figure 10.2* montre l'usage du CPC464 pour tracer trois graphiques en même temps, et cela à une vitesse raisonnable. La plage de valeurs de X, à la ligne 30, est fixée pour couvrir toute la largeur de l'écran, et, pour chaque valeur de X, trois valeurs de Y sont calculées. L'une est obtenue par le sinus de l'angle dont la valeur est X/639, en radians. Une autre valeur provient du carré du sinus de cet angle, et la troisième est son cube. Chaque point est marqué par les instructions PLOT des lignes 40, 60 et 80, en utilisant des couleurs d'INK différentes pour chaque courbe.

La raison des nombres utilisés dans les formules est que nous voulons que les courbes occupent tout l'écran. Le sinus

```
10 MODE 0
20 INK 0,0
30 FOR X=1 TO 639
40 Y=200+200*SIN(2*PI*X/639)
50 PLOT X,Y,1
60 Y=200+200*(SIN(2*PI*X/639)^2)
70 PLOT X,Y,2
80 Y=200+200*(SIN(2*PI*X/639)^3)
90 PLOT X,Y,3
100 NEXT
110 GOTO 110
```

Fig. 10.2. Un programme dessinant en haute résolution. Trois tracés séparés sont dessinés, en différentes couleurs.

d'un angle a une valeur comprise entre —1 et +1. Mais une valeur de 1 sur un axe Y n'est pas très manifeste, aussi multiplions-nous la valeur par 200. Cela fait varier la quantité entre —200 et 200 ; comme on ne peut marquer de point —200 par PLOT sur cet écran, on ajoute 200 à chaque valeur, ce qui donne des valeurs variant de 0 à 400, la plage complète des valeurs de Y.

L'autre point notable est qu'on utilise la formule 2*PI*X/639 pour l'angle. C'est pour permettre une mesure en radians. Ces fonctions trigonométriques ont une plage de valeurs pour un angle qui va de zéro radian à 2*PI radians. Quand X=0, 2*PI*X/639 vaut zéro, ce qui nous donne zéro radian à l'extrémité gauche de la courbe. Quand X=639, à l'extrémité droite de la courbe, l'angle est 2*PI*639/639 c'est-à-dire 2*PI radians, ce que nous voulons.

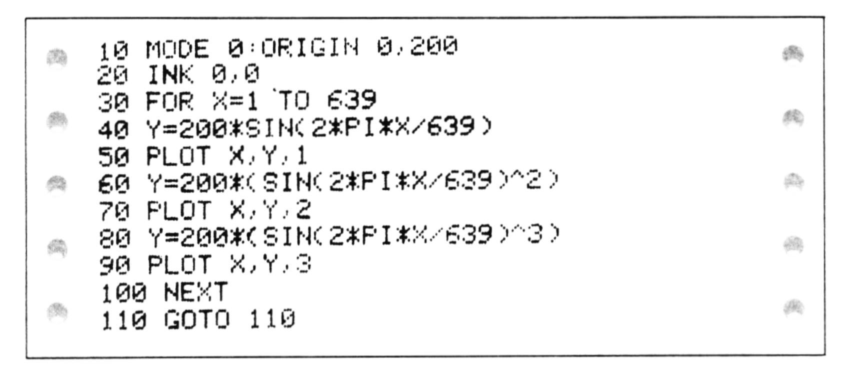
Si vous n'aimez pas travailler en radians, vous n'avez qu'à mettre l'instruction DEG sur une ligne avant l'usage des fonctions trigonométriques. Après l'exécution de DEG, tous les angles seront mesurés en degrés. Pour remettre la machine en radians, réinitialisez la machine, ou utilisez RAD.

Maintenant, après l'exécution du programme, examinez les courbes. Elles vous montrent très efficacement la taille du pixel en mode 0. Elles montrent aussi que la « basse résolution » du CPC464 a meilleur aspect que la haute résolution de certains ordinateurs ! Essayez de modifier la ligne 10, d'abord en MODE 1, puis en MODE 2, pour voir la petite taille des

pixels dans les autres modes. Il est moins facile de voir la couleur, mais les lignes de la courbe sont plus régulières et mieux jointes qu'en mode 0.

Vous ne verrez que deux courbes en mode 2. Pourquoi ? Parce que vous ne disposez que de deux couleurs en mode 2 ! Pour la plupart des usages, le mode 1 est le compromis idéal entre le nombre de couleurs et la haute résolution.

Nous n'avons pas encore tout à fait fini avec le tracé de courbes. Essayez le programme légèrement modifié de la *figure 10.3.*



```
10 MODE 0:ORIGIN 0,200
20 INK 0,0
30 FOR X=1 TO 639
40 Y=200*SIN(2*PI*X/639)
50 PLOT X,Y,1
60 Y=200*(SIN(2*PI*X/639)^2)
70 PLOT X,Y,2
80 Y=200*(SIN(2*PI*X/639)^3)
90 PLOT X,Y,3
100 NEXT
110 GOTO 110
```

Fig. 10.3. Utilisation d'ORIGIN pour décaler l'origine des coordonnées graphiques. C'est l'utilisation la plus simple d'ORIGIN.

Celui-ci commence par ORIGIN 0,200. L'effet de cette instruction est de décaler l'origine de toutes les courbes au point 0,200, le milieu gauche de l'écran. Toutefois l'origine est toujours prise comme le point obtenu avec 0,0, aussi PLOT 0,0 placera-t-il désormais un point à mi-hauteur de l'écran, sur la gauche.

Décaler la position de l'origine ainsi nous permet de simplifier les instructions graphiques, parce que nous n'avons plus besoin de la partie " + 200". On peut désormais écrire des choses comme PLOT 50,—50, parce que —50 signifie simplement 50 pixels en dessous du milieu, tout comme + 50 signifie 50 pixels au-dessus. L'instruction ORIGIN peut aussi être utilisée pour spécifier la taille d'une fenêtre, comme le montre le manuel.

Tracer des lignes

Tracer des courbes est un aspect très intéressant du graphique à haute résolution, mais on peut faire usage d'instructions qui vont beaucoup plus loin. Deux de ces instructions sont MOVE et DRAW.

L'instruction MOVE, comme son nom anglais l'indique — déplacer — déplace le curseur graphique. Mais comme le curseur graphique est invisible, vous ne voyez rien arriver quand vous utilisez MOVE. C'est comme déplacer un pinceau, mais en levant le pinceau du papier.

L'instruction DRAW est faite pour tracer une ligne — cette fois le pinceau est bien sur le papier. Chacune de ces instructions fait usage du même système de nombres (de « coordonnées ») que vous avez rencontré avec PLOT. De plus l'instruction DRAW peut utiliser un troisième nombre, le numéro d'encrier.

```
10 ON BREAK GOSUB 1000
20 MODE 0
30 MOVE 150,100
40 DRAW 150,300
50 DRAW 450,300
60 DRAW 450,100
70 DRAW 150,100
80 GOTO 80
1000 MODE 1
1010 LIST
```

Fig. 10.4. Un programme de dessin utilisant DRAW pour faire un carré.

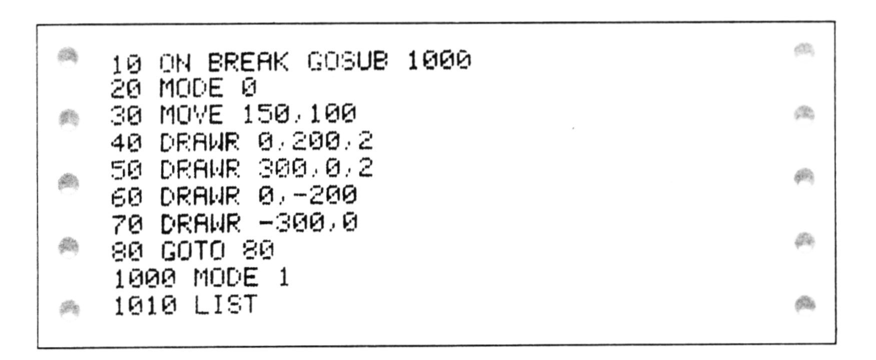
Il est temps de regarder un exemple, à la *figure 10.4.* Ce n'est rien de compliqué, un simple programme qui trace un carré. La couleur du carré sera « jaune d'or »sur un fond bleu si vous venez de mettre en route la machine, mais il sera probablement rouge sur noir si vous exécutez ce programme juste après le programme des courbes.

La raison en est que si vous ne donnez aucun numéro d'encrier après une instruction DRAW X,Y, la couleur qui sera utilisée sera la dernière utilisée dans une instruction

DRAW (ou PLOT). Ces machines ont de la mémoire! Vous verrez aussi que le carré est tracé très rapidement, plus vite que votre œil ne peut le suivre. Comparez ceci avec le temps que d'autres machines mettent, même pour un dessin aussi simple.

Ah oui, j'ai glissé une autre nouvelle instruction dans le programme. La ligne 10 utilise ON BREAK GOSUB 1000. Cela signifie que si vous tapez deux fois sur ESC (ce qui interrompt le programme par un BREAK), le sous-programme commençant à la ligne 1000 s'exécutera. J'ai utilisé cela parce que les listings n'ont pas bon aspect en mode 0. Le « sous-programme » à la ligne 1000 revient au mode 1, puis liste le programme. Comme une commande LIST revient *toujours* à "Ready", il ne sert à rien de mettre un RETURN à ce sous-programme. Quand vous appuyez deux fois sur ESC pour sortir de la boucle infinie à la ligne 80, l'écran revient en mode 1, et le programme est listé. C'est une chose utile à faire quand vous faites exécuter un programme.

Tant que nous en sommes à du dessin simple, autant voir comment on utilise DRAWR. Comme je l'ai dit plus tôt, cela veut dire DRAW avec des *coordonnées relatives.* DRAWR 10,100, par exemple, signifie DRAW, tracer une ligne 10 pixels sur la droite et 100 vers le haut. Ce *n*'est *pas* la même chose que tracer une ligne jusqu'au point 10,100, ce qui serait l'effet de DRAW tout court.



```
10 ON BREAK GOSUB 1000
20 MODE 0
30 MOVE 150,100
40 DRAWR 0,200,2
50 DRAWR 300,0,2
60 DRAWR 0,-200
70 DRAWR -300,0
80 GOTO 80
1000 MODE 1
1010 LIST
```

Fig. 10.5. Utilisation de DRAWR pour tracer un carré. Notez la différence dans les nombres.

La *figure 10.5* illustre notre tracé de carré avec DRAWR cette

fois. Si vous voulez un déplacement vers la droite ou le haut, les distances sont positives. Si vous voulez aller vers la gauche ou le bas, les distances sont négatives. Si vous voulez garder une coordonnée, X ou Y, identique, la valeur à utiliser est 0.

```
10 MODE 1
20 CLG
30 FOR N=1 TO 50
40 MOVE 320,200
60 DRAW RND(1)*639,RND(1)*399,RND(1)*4
70 NEXT
80 AFTER 200 GOSUB 1000
90 GOTO 90
1000 CLG
1010 MOVE 320,200
1020 FOR N=1 TO 50
1030 A=RND(1):IF A>0.5 THEN A=1 ELSE A=-
1
1040 DRAWR A*INT(RND(1)*100),A*INT(RND(1
)*100),RND(1)*4
1050 NEXT
1060 RETURN
```

Fig. 10.6. Motifs linéaires aléatoires, illustrant les différences entre DRAW et DRAWR.

Juste pour bien intégrer les différences entre DRAW et DRAWR, la *figure 10.6* montre un ensemble de lignes tracées du centre de l'écran jusqu'à des positions aléatoires, en utilisant DRAW (ligne 60). Après une pause, un effet tout différent est réalisé à l'aide de DRAWR. Cette fois chaque nouvelle ligne est attachée à la fin de la précédente. En utilisant la ligne 1030 pour engendrer une valeur de A qui soit 1 ou —1, la direction de chaque ligne est aléatoire, et sa longueur est aussi rendue aléatoire par la ligne 1040. Cela produit un motif connu sous le nom de « cheminement aléatoire » — on dirait le trajet d'une mouche affolée.

La pause a été organisée grâce à une nouvelle instruction, AFTER 200 GOSUB 1000. Elle signifie qu'après un compte de 200 sur le compteur de l'horloge, le programme exécutera le sous-programme de la ligne 1000. Comme auparavant, l'horloge compte à un rythme de 50 par seconde, si bien que 200

représente 4 secondes. Après ce temps, le GOSUB 1000 fait la routine DRAWR, et puis revient à la boucle infinie de la ligne 90.

Comment s'organiser

C'est très bien de dessiner en haute résolution, mais pour travailler minutieusement, il vous faut préparer quelques plans. La meilleure aide à la planification est une feuille de papier millimétré. Le format A4 est idéal. Si vous prenez quatre centimètres pour 100 pixels, vous pouvez marquer des graduations de 0 à 400 dans la largeur de la feuille, et de 0 à 650 dans sa hauteur. Un quart de tour pour que le côté le plus long représente la largeur de l'écran, et vous pouvez préparer vos dessins directement sur cette feuille ou en utilisant du papier calque attaché avec un trombone. Quand votre dessin (limitons-nous pour l'instant à de simples traits) est complet, marquez les points d'intersection des segments de droite, et notez les coordonnées de ces points. Ensuite vous pouvez commencer à écrire votre programme.

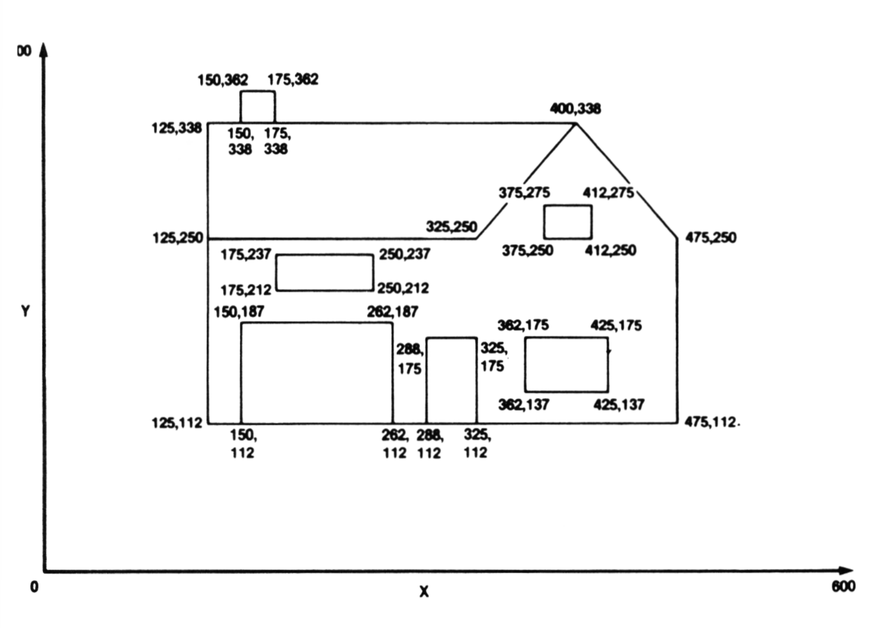


Fig. 10.7. Plan de tracé d'une maison, obtenu sur un calque par-dessus du papier millimétré.

La *figure 10.7* illustre comment préparer un dessin selon cette méthode. La maison a été dessinée sur du papier calque, avec le papier millimétré par-dessous. L'esquisse grossière a été redessinée pour suivre les lignes du papier millimétré, et les coordonnées de chaque point ont été lues sur le papier millimétré.

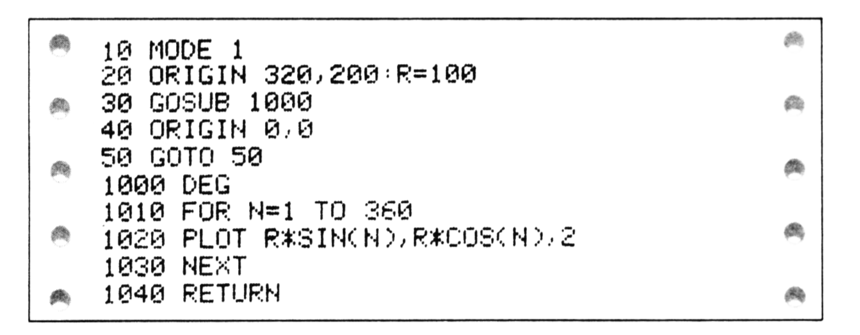
```
10 MODE 0
20 MOVE 125,112
30 PEN 2
40 FOR N=1 TO 7
50 READ X,Y
60 DRAW X,Y
70 NEXT
80 MOVE 150,112:FOR N=1 TO 3
90 READ X,Y:DRAW X,Y:NEXT
100 MOVE 288,112
110 FOR N=1 TO 3
120 READ X,Y:DRAW X,Y:NEXT
130 MOVE 362,137
140 FOR N=1 TO 4:READ X,Y
150 DRAW X,Y:NEXT
160 MOVE 375,250
170 FOR N=1 TO 4:READ X,Y
180 DRAW X,Y:NEXT
190 MOVE 175,212
200 FOR N=1 TO 4:READ X,Y
210 DRAW X,Y:NEXT
220 MOVE 150,338
230 FOR N=1 TO 3:READ X,Y
240 DRAW X,Y:NEXT
250 MOVE 125,338:DRAW 400,338
1000 DATA 475,112,475,250,400,338,325,25
0,125,250,125,338,125,112
1010 DATA 150,187,262,187,262,112
1020 DATA 288,175,325,175,325,112
1030 DATA 362,175,425,175,425,137,362,13
7
1040 DATA 375,275,412,275,412,250,375,25
0
1050 DATA 175,237,250,237,250,212,175,21
2
1060 DATA 150,362,175,362,175,338
```

Fig. 10.8. Le programme qui trace la maison.

Cela mène au programme de la *figure 10.8.* Celui-ci utilise autant que possible des lignes de DATA, pour que toute éventuelle amélioration du dessin soit limitée à une modification des lignes de données. Chaque opération de dessin d'une partie commence par une instruction MOVE. Je commence toujours par déplacer le curseur au coin inférieur gauche de ce que je dessine. Après le MOVE, un ensemble d'instructions DRAW dans une boucle effectue le tracé. C'est très rapide et efficace, mais le temps que vous y passeriez sans préparation serait criminel !

Faire des cercles

Les dessins qui ne comportent que des traits droits sont très bien, mais pour beaucoup d'usages, on a besoin de cercles, d'ellipses et d'arcs. C'est l'une des rares faiblesses du CPC464, il n'a pas d'instruction CIRCLE, ni d'instruction de remplissage en couleur d'un tracé. Il faut s'y résigner et utiliser des sous-programmes personnels pour effectuer ces tâches. Le manuel est très utile et explicite à ce sujet.

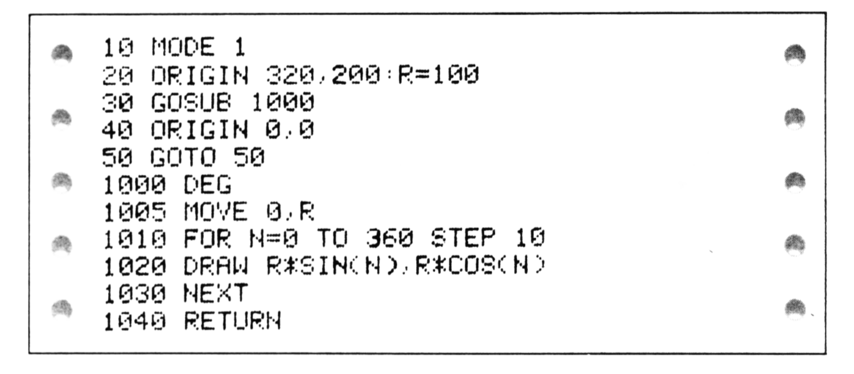


```
10 MODE 1
20 ORIGIN 320,200:R=100
30 GOSUB 1000
40 ORIGIN 0,0
50 GOTO 50
1000 DEG
1010 FOR N=1 TO 360
1020 PLOT R*SIN(N),R*COS(N),2
1030 NEXT
1040 RETURN
```

Fig. 10.9. Un programme de tracé de cercle. La méthode est plutôt lente.

La *figure 10.9* montre un programme de tracé de cercle. La ligne 20 fixe le mode, pour le cas où il aurait été modifié auparavant, et cela nettoie l'écran. La ligne 20 fixe ensuite le centre du cercle comme origine, et détermine le rayon, R. Le sous-programme de tracé de cercle est ensuite appelé, puis le programme redéplace l'origine, et fait une boucle infinie jusqu'à ce que vous appuyiez sur ESC deux fois. Dans le sous-

programme, DEG fixe la mesure des angles en degrés. La ligne 1010 commence ensuite une boucle, marquant chaque point sur le périmètre du cercle. Si vous avez fait — et que vous vous en souveniez — de la géométrie analytique à l'école, vous comprendrez facilement ce qui se passe. Si ce n'est pas le cas, prenez la routine pour acquise plutôt que de vous mêler de mathématiques, le sous-programme trace effectivement un cercle.



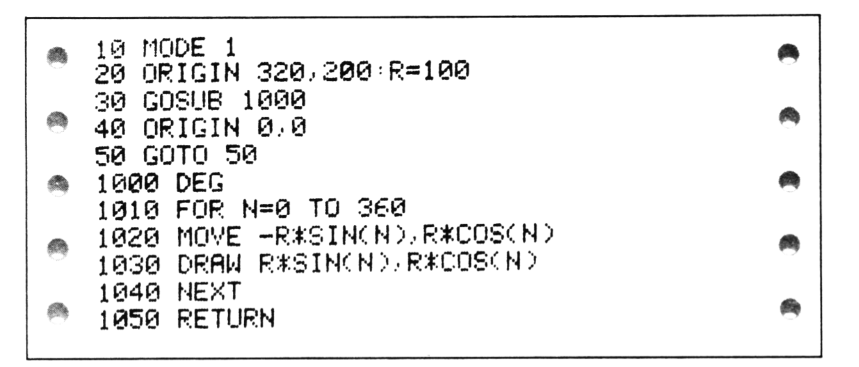
```
10 MODE 1
20 ORIGIN 320,200:R=100
30 GOSUB 1000
40 ORIGIN 0,0
50 GOTO 50
1000 DEG
1005 MOVE 0,R
1010 FOR N=0 TO 360 STEP 10
1020 DRAW R*SIN(N),R*COS(N)
1030 NEXT
1040 RETURN
```

Fig. 10.10. Une méthode bien plus rapide pour tracer un cercle.

Ce n'est pas la seule méthode pour tracer un cercle, et elle est très lente, mais au moins elle est assez simple. La *figure 10.10* présente une solution différente, beaucoup plus rapide. Elle ne trace pas un vrai cercle, mais une figure faite d'un grand nombre de côtés (36), cependant, la forme tracée se rapproche raisonnablement d'un cercle, et la vitesse est un argument de poids ! Ce sous-programme doit être adopté si vous ne voulez pas attendre pendant un tracé de cercle

Tant que vous travaillez avec les cercles, il est bon de voir rapidement comment remplir un cercle d'une couleur donnée. La *figure 10.11* vous en donne un avant-goût avec les lignes 10 et 20 qui fixent l'origine et le rayon, et le sous-programme qui commence en 1000 qui trace et remplit le cercle. Le fait que les lignes soient en couleur revient à remplir le cercle de couleur.

L'effet de remplissage est réalisé aux lignes 1020 et 1030. L'instruction MOVE envoie à un ensemble de points sur le



```
10 MODE 1
20 ORIGIN 320,200:R=100
30 GOSUB 1000
40 ORIGIN 0,0
50 GOTO 50
1000 DEG
1010 FOR N=0 TO 360
1020 MOVE -R*SIN(N),R*COS(N)
1030 DRAW R*SIN(N),R*COS(N)
1040 NEXT
1050 RETURN
```

Fig. 10.11. Comment remplir un cercle de couleur. Les formes rectangulaires sont bien plus faciles à remplir.

côté gauche du cercle, et puis DRAW trace une ligne jusqu'au point correspondant sur le côté droit. Essayez, c'est bon visuellement, et presque aussi rapide que l'instruction FILL de certains ordinateurs.

Jouer au détective : la position du curseur graphique

On peut obtenir beaucoup d'effets intéressants si le programme détecte automatiquement la position du curseur. Souvenez-vous que la position du curseur graphique est invisible, aussi la détection de sa position ou de son dernier tracé doit-elle être confiée à une instruction.

Il y en a en fait plusieurs, qui rapportent ce que fait le curseur. Deux d'entre elles sont XPOS et YPOS. Chacune renvoie un nombre. Si vous utilisez J=XPOS, la valeur de J sera la coordonnée X du curseur au moment du test. YPOS fournit évidemment la coordonnée Y du même curseur. Ces instructions sont analogues aux instructions POS et VPOS du curseur texte, mais pour le curseur graphique. La différence est que XPOS et YPOS ne sont pas suivis de numéro de fenêtre. Il nous faut maintenant regarder comment utiliser ces instructions.

On peut utiliser ces instructions à une tâche très utile, nous assurer que le curseur graphique ne sorte jamais de l'écran.

On peut par exemple faire du « réenroulement ». Cela signifie que si le curseur sort de l'écran sur un côté, il réapparaît de l'autre.

```
10 CLS
20 X=320:Y=200
30 MOVE X,Y:RANDOMIZE TIME
40 WHILE X<>0
50 J=RND(1):IF J>0.5 THEN X=20*RND(1) EL
SE Y=20*RND(1)
60 DRAWR X,Y,1
70 IF XPOS>630 THEN MOVE 10,YPOS
80 IF XPOS<10 THEN MOVE 630,YPOS
90 IF YPOS>390 THEN MOVE XPOS,10
100 IF YPOS<10 THEN MOVE XPOS,390
110 WEND
```

Fig. 10.12. Utilisation d'XPOS et YPOS pour réaliser un « réenroulement ».

La *figure 10.12* illustre cela, dans un programme qui trace des lignes brisées. La ligne 20 fixe une position au milieu de l'écran, et la ligne 30 y amène le curseur. La ligne 30 contient aussi l'instruction RANDOMIZE TIME. La raison de ce point est que RND n'engendre pas de « vrais » nombres aléatoires, en tout cas rien d'aussi aléatoire que les nombres d'une roulette de casino. Les nombres fournis par RND sont calculés par l'ordinateur, et comme tout ce qui est calculé par une formule, ils ne peuvent être vraiment imprévisibles. Si bien que les nombres fournis par RND, si vous en affichez suffisamment, vont se mettre à répéter leur séquence.

Vous pouvez interrompre cela en utilisant RANDOMIZE, et le résultat est encore meilleur si le nombre qui suit RANDOMIZE est lui-même un nombre qui a peu de chance de se répéter. TIME est un bon candidat, aussi RANDOMIZE TIME est-il un moyen vraiment sûr pour éviter d'avoir deux fois le même affichage avec ce programme.

La ligne 40 commence ensuite une boucle infinie, car X ne sera jamais 0. La ligne 50 décide ensuite des nouvelles valeurs de X ou Y. L'une ou l'autre est changée en un nombre aléatoire, suivant la valeur de J. A la ligne 60, les nouvelles valeurs de X et Y sont utilisées pour tracer une ligne, mais

avec des valeurs relatives, et non absolues. On teste ensuite la position du curseur aux lignes 70 à 100. Ces tests vérifient que si le curseur touche la limite de l'écran, il réapparaîtra sur le côté opposé.

Notez l'usage de XPOS et de YPOS dans la partie MOVE de ces lignes. Vous ne pouvez utiliser X et Y, parce que ce sont des positions relatives, et non absolues.

Il y a un autre moyen de vérifier où se trouve le curseur graphique invisible. L'instruction TEST(X,Y) donne un numéro d'INK, qui est le numéro d'INK de la position X,Y sur l'écran. Vous pouvez donc mettre TEST(X,Y) dans la partie conditionnelle d'un IF...THEN pour trouver si tel point de l'écran est de la couleur du fond, ou d'un obstacle. Vous pouvez aussi utiliser TESTR, qui prend des coordonnées relatives. Utiliser TESTR(1,0), par exemple, testera le pixel juste à droite du curseur en mode 2. Vous pouvez utiliser TESTR(2,0) en mode 1, et TESTR(4,0) en mode 0, pour le même objectif.

En suivant le mouvement : l'animation en mode graphique

Nos graphiques à haute résolution ont jusqu'ici été plutôt statiques, restreints à des tracés de motifs ou de formes. Cela n'a rien d'obligé, parce qu'il existe une instruction très utile, TAG. TAG relie le curseur graphique au curseur texte. Maintenant le curseur graphique est petit, il a pour taille un seul pixel, et le curseur texte fait 8 × 8 pixels. Le point d'attache est le sommet gauche du curseur texte. Si vous affichez par PRINT un caractère à la position donnée par le curseur graphique, donc, le caractère occupera un espace de sept pixels à droite du curseur graphique, et sept en dessous. Je pense à un caractère de texte construit sur 7 × 7 pixels. Vous pouvez bien sûr utiliser un caractère « redéfini » de 8 × 8.

L'effet de TAG est que l'on peut afficher des motifs comme ceux obtenus par un CHR$ sur l'écran *à la position du curseur graphique*. Quelle importance ? Eh bien, si vous pensez à l'écran du mode 1, par exemple, nous n'avons que 40 positions pour un caractère sur une ligne. Si nous voulons déplacer des motifs sur la ligne, nous devrons les déplacer en quarante mouvements seulement, ce qui fera une animation très

saccadée. Le curseur graphique, lui, peut bouger selon des pas bien plus fins. En MODE 1, le curseur graphique peut marquer 320 pas, en utilisant des valeurs de X de 0 à 639. C'est un mouvement bien plus petit, et il peut faire une bien meilleure animation.

```
10 MODE 1
20 Y=200
30 TAG
40 FOR X=0 TO 639 STEP 2
50 MOVE X,Y
60 PRINT " ";
70 MOVER 2,0
80 PRINT CHR$(243);
90 CALL &BD19
100 NEXT
110 TAGOFF
120 END
```

Fig. 10.13. Utilisation de TAG pour faire apparaître des motifs graphiques à la position du curseur.

Prenez comme échantillon la *figure 10.13*. L'instruction TAG de la ligne 30 effectue cette tâche d'attacher le curseur texte. Quiconque a programmé sur le micro BBC reconnaîtra cet effet, c'est l'équivalent de l'instruction VDU5 du BBC. On peut maintenant afficher à la position du curseur graphique par PRINT. La ligne 40 détermine une simple boucle qui fera traverser l'écran au curseur graphique. La ligne 50 déplace le curseur graphique à la position fixée par X et Y.

Notez le pas de 2 dans la boucle. C'est parce que la boucle prend la plage usuelle de 0 à 639 pour contrôler le curseur, mais en mode 1 seules 320 positions sont distinguées. Chaque position du curseur graphique peut avoir deux valeurs de X. X=0 et X=1 définissent la même position, la suivante est X=2 ou X=3, et ainsi de suite. En MODE 2, chaque nombre de 0 à 639 désigne une position séparée — essayez le programme dans ce mode. En MODE 0 il n'y a que 160 positions séparées sur l'écran, aussi chacune d'elles peut-elle être désignée par quatre valeurs de X ; X=0, 1, 2 ou 3 donne la première position, et X=4, 5, 6 ou 7 la deuxième, etc. Dans une boucle, on peut utiliser STEP 4.

Ces valeurs de pas sont importantes, car elles permettent d'éliminer une autre cause d'irrégularité dans le mouvement. Si le motif (en mode 0) effectue quatre passages de boucle sur chaque position avant de se déplacer, le mouvement sera beaucoup plus saccadé que si la forme bouge à chaque tour de boucle. Ensuite, on affiche un espace à la place du curseur. L'objet de cette opération est de supprimer l'ancien dessin, ce qui est sans effet au premier passage. La ligne 70 déplace ensuite le curseur de deux pas, et la ligne 80 affiche le caractère motif. C'est à ce point qu'il faut être attentif. Essayez d'exécuter le programme sans le point-virgule, et regardez les étranges résultats ! Quand vous utilisez l'écran graphique, *tout s'affiche,* y compris le retour chariot et le saut de ligne. Quand vous utilisez TAG, vous pouvez supprimer ces codes grâce au point-virgule, qui n'envoie pas de code.

Le mystère suivant se trouve à la ligne 90. Cela implique un peu de « code machine ». Pour voir à quoi cela sert, le mieux est de supprimer cette ligne, et d'exécuter le programme. Vous verrez la forme se déplacer plus vite, mais avec des effets bizarres par moments. La raison de ces curieuses distorsions du motif est que l'écran TV forme les images en décrivant les lignes horizontales, de haut en bas et en rendant brillants les points où quelque chose doit apparaître à l'écran. Quand l'objet sur l'écran se déplace lui aussi, il peut arriver qu'il y ait des conflits entre les deux mouvements. Pour éviter ce problème, on doit déplacer l'objet à une vitesse synchronisée avec le rythme du spot lumineux de l'écran. Il y a une routine de la MEM du CPC464 qui fait cela, et on peut l'appeler par CALL &BD19. "&BD19" est en fait un nombre, écrit en code *hexadécimal (base 16).*

Évidemment, vous pouvez trouver que ralentir le mouvement est la dernière chose à faire. Comment l'accélérer au contraire ? On peut déjà transformer X et Y en entiers, X% et Y%. Si cela ne vous suffit pas, essayez STEP 4 ou même STEP 8. Ne tentez pas d'omettre le CALL &BD19, car les défauts de synchronisation seront d'autant plus visibles que l'objet se déplace plus vite. Notez que dans ce programme, l'instruction MOVER de la ligne 70 sert à garder une partie de la forme sur l'écran, ce qui contribue à atténuer les saccades. Avec d'autres types de motifs, vous pourrez être amené à omettre cette instruction, ou à utiliser un nombre différent.

Animation avec INK

Une tout autre forme d'animation, très astucieuse, repose sur l'usage de l'instruction INK. Supposez que ce que vous voulez animer n'est pas un motif en CHR$ (qui peut être, souvenez-vous-en, un motif redéfini par vous), mais un dessin complet, créé par DRAW et MOVE. Vous pourriez bien sûr dessiner la forme, attendre, l'effacer, la redessiner à un emplacement différent, etc. Le problème est que ce serait lent, et que l'animation serait saccadée.

Une bien meilleure méthode consiste à utiliser les incroyables instructions INK. Supposons que vous fassiez un ensemble de dessins, chacun dans une position différente, et chacun avec une encre tirée d'un encrier différent. Supposez aussi que l'encre de chaque pot soit la couleur du fond. Le résultat, évidemment, est que tous vos dessins sont invisibles !

Maintenant imaginez une boucle dans votre programme. A chaque passage de la boucle, vous changez une des couleurs d'encrier que vous avez utilisées pour une nouvelle couleur, qui ne soit pas celle du fond. Cela aura pour effet de rendre un de vos dessins visible. Attendez un moment, et remettez l'encrier de la couleur du fond. Le dessin disparaît, et vous pouvez changer l'encre d'un autre encrier pour faire apparaître un autre dessin, puis le faire disparaître. Ainsi, vous pouvez faire un ensemble de dessins *mis au préalable sur l'écran,* et les faire apparaître ou disparaître tour à tour, en donnant une illusion d'animation. Le meilleur effet est évidemment obtenu en mode 0, parce que vous pouvez jouer sur bien plus d'encriers.

La *figure 10.14* montre ce que j'ai dans l'esprit. La première partie de ce programme met dans les encriers 2 à 15 la couleur du fond (la couleur 1). Les lignes 40 à 90 tracent ensuite quatre rectangles sur l'écran, le premier est vertical, et les trois suivants sont décalés chacun de 45 degrés par rapport au précédent. Le tout montre les positions successives d'un rectangle qui passe de la position verticale à la position horizontale, et inversement, dans le sens des aiguilles d'une montre. Si vous voulez examiner le détail des événements, mettez le numéro d'INK de la ligne 30 à 24, au lieu de 1, et mettez STOP à la ligne 95.

```
10 MODE 0
20 FOR CL=2 TO 15
30 INK CL,1:NEXT
40 FOR NR=1 TO 4
50 READ X,Y:MOVE X,Y
60 FOR LN=1 TO 4
70 READ X,Y
80 DRAW X,Y,NR+1
90 NEXT:NEXT
100 WHILE INKEY(47)=-1
110 FOR NR=2 TO 5
115 CALL &BD19
120 INK NR,24
130 FOR X=1 TO 30:NEXT
140 INK NR,1
150 NEXT
160 WEND
170 DATA 300,300,350,300,350,100,300,100
,300,300
180 DATA 385,288,415,252,270,113,235,148
,385,288
190 DATA 225,225,425,225,425,175,225,175
,225,225
200 DATA 230,255,275,293,410,148,377,110
,230,255
210 MODE 1:END
```

Fig. 10.14. Animation réalisée en utilisant de l'encre « invisible » !

Une fois que les dessins sont faits, avec l'écran encore vide, vous pouvez les faire apparaître en changeant les couleurs d'INK. La boucle qui commence à la ligne 100 se poursuivra jusqu'à un appui sur la barre d'espacement. La boucle FOR...NEXT change ensuite chaque couleur d'INK tour à tour, pour un court instant, puis revient à la couleur du fond. Le CALL &BD19 est ajouté pour rendre l'animation plus souple — essayez de supprimer l'instruction pour voir la différence. L'impression globale est celle d'un rectangle qui tourne dans le sens des aiguilles d'une montre.

En graphique, on n'arrive à rien sans un peu de sueur et de peine. Les coordonnées des différents dessins doivent être déterminées, et mises dans des lignes de DATA. Cela implique un travail fastidieux, qui est plus facile si vous avez une tablette graphique ! Il est très probable qu'un de ces jours

quelqu'un élaborera un programme qui fait les mesures pour vous. Vous produirez le dessin, et le programme vous donnera les coordonnées de chaque point quand la forme est tournée selon un angle spécifié par vous. C'est un problème de géométrie que l'ordinateur peut résoudre bien plus vite que vous. Cette méthode d'animation, incidemment, est aussi utilisée sur le micro BBC, mais avec moins de simplicité dans la mise en œuvre.

11

Les capacités sonores du CPC464

La capacité de produire des sons est un trait essentiel de tous les ordinateurs modernes. Le son du CPC464 provient de son haut-parleur intégré, qui peut être réglé en volume à l'aide d'un bouton situé sur le flanc droit de l'ordinateur. De plus une prise au dos du CPC464 vous permet de le relier à un amplificateur, si bien que vous pouvez donner plus de volume au son, et obtenir de meilleurs graves. La différence est tout à fait étonnante si vous n'avez jamais entendu que le son du petit haut-parleur de l'ordinateur. En plus, le son issu de la prise peut être en stéréo.

Si vous avez des écouteurs de baladeur standard, vous pouvez les brancher à cette prise, à l'extrémité droite au dos du CPC464. Le volume sonore n'est pas très fort sur les écouteurs, et un amplificateur stéréo est nécessaire pour obtenir le meilleur effet.

Maintenant si vous vous contentez d'un simple bip dans vos programmes, pour vous signaler quoi que ce soit, comme faire un choix, vous n'avez pas besoin d'aller plus loin que PRINT CHR$(7). Cela vous donnera votre bip, et la *figure 11.1* vous fournit une application d'une pareille note. C'est le début d'un programme imaginaire de démonstration sonore ; ce qui nous intéresse est le sous-programme qui commence à la ligne 1000. Celui-ci utilise K$=INKEY$ pour détecter une touche, et si une touche est enfoncée, J=VAL(K$) donne sa valeur numérique utilisée dans le reste du programme.

```
10 CLS:PRINT TAB(19)"MENU"
20 PRINT:PRINT TAB(3)"1. MUSIQUE"
30 PRINT TAB(3)"2. EXPLOSIONS"
40 PRINT TAB(3)"3. BRUIT D'EAU"
50 PRINT TAB(3)"4. MOTEURS"
60 PRINT:PRINT"Choisissez un nombre, S.V
.P."
70 GOSUB 1000
80 IF J<1 OR J>4 THEN PRINT "Incorrect -
 Essayez à nouveau, S.V.P.":GOTO 10
90 ON J GOSUB 500,500,500,500
100 END
500 PRINT "Vous Pourrez écrire ici un Pr
ogramme,":PRINT "Plus tard."
510 RETURN
1000 K$=INKEY$:IF K$<>"" THEN J=VAL(K$):
RETURN
1010 PRINT CHR$(7);:FOR N=1 TO 500:NEXT
1020 GOTO 1000
```

Fig. 11.1. Production d'un bip avec CHR$(7).

Le point intéressant est ce qui se passe si aucune touche n'est enfoncée. La ligne 1010 fait entendre un court bip, grâce à CHR$(7). Vous devez mettre un point-virgule après CHR$(7), sans quoi l'écran se déroulera. Bien que PRINT CHR$(7) n'affiche rien sur l'écran, le curseur sautera une ligne après chaque PRINT, et si vous attendez quelque temps avant d'appuyer sur une touche, vous verrez l'écran se dérouler. Enlevez le point-virgule pour voir ce que je veux dire. Après le bip, il y a une courte pause, et le GOTO 1000 renvoie au test d'INKEY$.

Le résultat global est que la machine vous corne aux oreilles jusqu'à ce que vous appuyiez sur une touche. C'est une solution différente de celle de l'astérisque clignotant. Vous pouvez même combiner les deux procédés si vous voulez, pour être bien sûr que personne ne peut ignorer le choix sur le menu !

Généralités sur le son

Dès que vous voulez dépasser le simple bip, vous devez en savoir un peu plus sur le son lui-même. Ce qu'on appelle

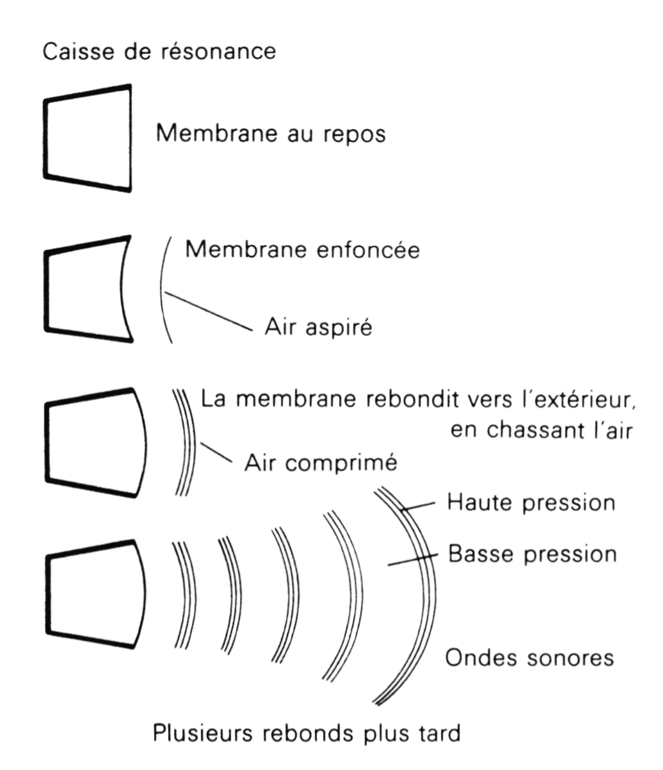


Fig. 11.2. **Comment une membrane de caisse de résonance produit des sons en compressant et décompressant alternativement de l'air.**

« son » est le résultat de rapides variations de pression dans l'air autour de nos oreilles. Tout ce qui engendre un son le fait en modifiant la pression de l'air, et la *figure 11.2* montre comment la membrane d'une caisse de résonance y parvient. Les variations de pression de l'air sont invisibles, et nos oreilles ne détectent même pas ces variations, sauf si les changements sont très rapides.

On mesure le rythme des changements en cycles par seconde, ou *hertz*. Le cycle d'une onde est un ensemble de changements de pression, d'abord dans une direction, puis dans l'autre, et enfin en retour à la normale, tel qu'on peut le représenter à la *figure 11.3*. On parle d'« onde » sonore parce que la forme de la courbe de pression est analogue aux ondulations d'un liquide. La courbe montre comment évolue la pression de l'air pendant la génération du son.

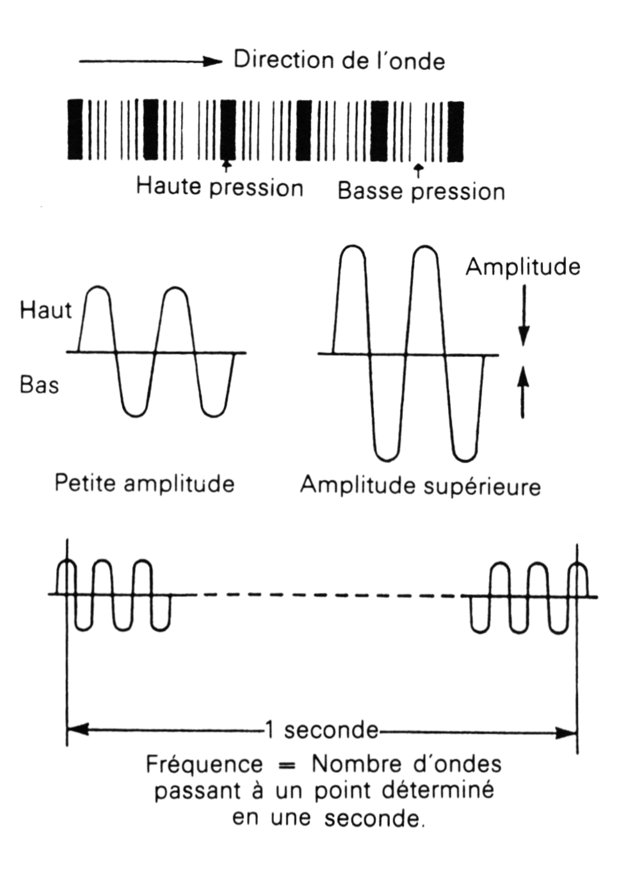


Fig. 11.3. **La fréquence et l'amplitude d'une onde sonore.**

La *fréquence* d'un son est caractérisée par son nombre de *hertz* — le nombre de cycles de changement de pression d'air par seconde. Si cette quantité fait moins de 20 hertz, nous ne pouvons entendre de son, bien que les variations soient réelles et puissent avoir des effets gênants. Nous pouvons entendre l'effet des ondes de pression dans l'air dans la plage de fréquence qui va de 20 hertz à environ 15000 hertz. La différence de fréquence des ondes correspond à ce que nous ressentons comme la « hauteur » d'une note. Une fréquence faible, de 80 à 120 hertz correspond à une note grave, de basse hauteur. Une fréquence de 4000 hertz ou plus correspond à une note aiguë, de fréquence haute.

L'amplitude du changement de pression détermine ce que l'on appelle la *force* d'une note. On la mesure en terme d'*amplitude*, c'est-à-dire le changement maximum de pression d'air par rapport à la valeur normale. Pour contrôler complètement la génération de sons, il faut pouvoir spécifier l'amplitude, la fréquence, la forme de l'onde, et aussi les modifications de l'amplitude de la note durant le temps où le son est audible.

Pour écrire de la musique, un compositeur doit spécifier pour chaque note sa force, sa longueur, et sa hauteur. Dans la notation musicale, la force est indiquée par des lettres comme *f* (fort) et *p* (doux), que l'on peut doubler ou tripler si nécessaire. Ainsi *fff* signifie très fort, et *ppp* très doucement. La durée de chaque note est indiquée de deux manières. L'une d'elles consiste en une mesure de métronome.

Symbole	Durée	Nom
𝅘𝅥𝅰	1/8	Triple croche
𝅘𝅥𝅯	1/4	Double croche
♪	1/2	Croche
♩	1	Noire
𝅗𝅥	2	Blanche
𝅝	4	Ronde

Fig. 11.4. Les symboles utilisés en musique pour indiquer la durée de chaque note.

Un métronome est un générateur de sons qui fait entendre un tic-tac mesurant des intervalles précis. Les mesures des métronomes indiquent combien de « battements » (unités de notes) se font entendre à la minute. L'unité de durée des notes est appelée la *noire*; ainsi la mesure du métronome décide de la durée d'une noire en marquant le nombre de noires qu'on peut faire entendre en une minute. Les durées des autres notes sont ensuite mesurées selon celle de la noire. Une *blanche* dure deux fois le temps d'une noire, et une *ronde* deux fois le temps d'une blanche, donc quatre fois une noire. La *croche* a la moitié du temps de la noire, et la *double-croche* le quart. On indique ces différentes durées de

Symbole de pause	Durée
𝅀	1/4
𝄾	1/2
𝄽	1
𝄼	2
𝄻	4

Fig. 11.5. Les symboles des silences en musique.

notes par la forme des symboles utilisés pour marquer les notes *(figure 11.4)*. De plus, il y a des symboles pour les différentes durées de silence en musique, comme cela est illustré sur la *figure 11.5*. Certaines partitions musicales ne précisent pas de mesure du métronome, mais font usage des termes italiens lento, moderato ou allegro, qui indiquent plus librement la durée de la noire.

La hauteur d'une note est indiquée dans la notation musicale par sa place sur une sorte de « grille » de préparation pour la musique qu'on appelle *portée*. La musique pour piano présente deux de ces portées, qui consistent chacune en cinq lignes et quatre intervalles. Les lignes marquées par le signe analogue à la esperluette (&), en fait la clé de sol, forment la portée aiguë, utilisée pour les notes les plus hautes, et la portée inférieure est la portée basse. La portée basse est marquée par une clé ressemblant à un C renversé.

La musique écrite pour des instruments sans clavier utilise une seule portée. La musique pour orgue et piano montre toujours deux portées, avec l'emplacement pour une note au milieu de l'intervalle qui les sépare. Cette note est appelée *do majeur* (*C* dans la notation anglo-saxonne), et, sur un piano, on joue cette note en appuyant sur la touche qui se trouve presque au milieu du clavier. La *figure 11.6* montre des portées avec des les notations *française* et anglo-saxonne des notes.

Les notes de la *figure 11.6* sont regroupées par huit. On appelle un pareil groupe une *octave*. Il est possible de jouer des notes de hauteur intermédiaire à celles de deux notes

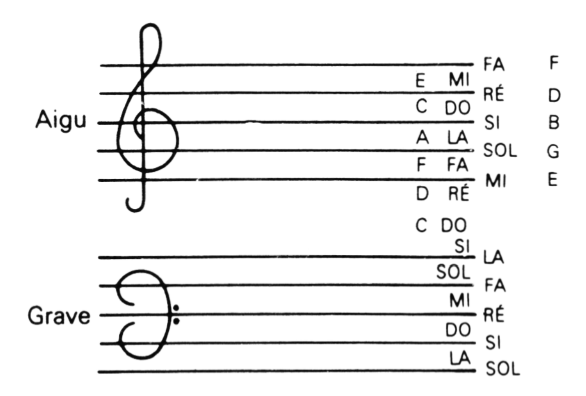


Fig. 11.6. Les portées musicales, avec les noms des notes. (Notez la position du DO central.)

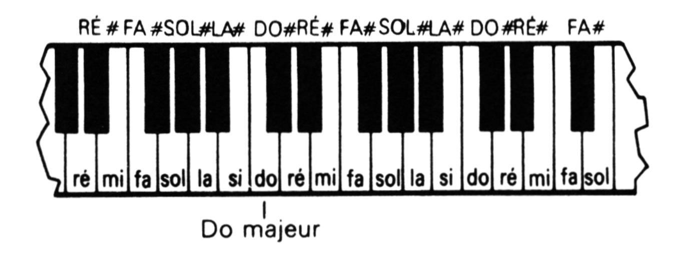


Fig. 11.7. Partie du clavier de piano, avec le DO majeur. Il n'y a qu'un demi-ton entre MI et FA, et entre SI et DO.

consécutives d'une octave. On appelle ces notes « intercalées » des *demi-tons.* La musique occidentale utilise traditionnellement un total de douze notes distinctes, tons et demitons, dans une octave. Cette gamme complète est représentée à la *figure 11.7,* qui montre l'aspect d'une partie de clavier de piano. Les demi-tons sont marqués sur le piano par les touches noires principalement ; mais comme, entre mi et fa et entre si et do, l'intervalle est d'emblée un demi-ton, il n'y a pas de touche noire intercalée.

Sur les partitions, les demi-tons sont indiqués par les signes # (dièse) et *b* (bémol). Un dièse indique que la hauteur doit être élevée d'un demi-ton au-dessus de la note écrite, et un bémol que la note doit être baissée d'un demi-ton. Sur le piano, le demi-ton au-dessus d'une note est la même note que

le demi-ton au-dessous de la note supérieure, ainsi do dièse est la même note que ré bémol. Ce n'est pas vrai de tous les instruments.

Le son du CPC464

Au travail, maintenant. Le CPC464 offre une instruction sonore qu'on peut utiliser plus ou moins simplement. Votre choix dépendra de vos besoins ; ainsi vous n'aurez pas grande préparation à faire pour produire une note d'avertissement, ou pour jouer une mélodie. Une fois que vous serez accoutumé à l'utilisation de l'instruction SOUND, cependant, vous pourrez l'utiliser de manière bien plus riche. Parmi tous les ordinateurs que je connais, le CPC464 est de loin celui qui a le meilleur système sonore, du point de vue du contrôle du son par des instructions BASIC. Nous allons commencer par la simple production de notes.

On produit des notes grâce à l'instruction SOUND. L'instruction SOUND doit être suivie d'au moins deux nombres. Le premier d'entre eux est un « numéro de canal ». Le CPC464 peut produire trois notes sonores à la fois, et chacune de ces trois notes peut être contrôlée indépendamment. On le fait en allouant chaque note à un canal séparé. Ces canaux sont étiquetés A, B, C, et on peut en sélectionner un ou plusieurs en utilisant le numéro de canal.

Numéro de choix	Canal choisi
1	A
2	B
3	A et B
4	C
5	A et C
6	B et C
7	A et B et C

Fig. 11.8. Les nombres utilisables pour choisir les canaux.

La *figure 11.8* montre comment on utilise les nombres de 1 à 7 pour choisir un ou plusieurs canaux. Le CPC464 va plus loin que n'importe quel micro-ordinateur à ce jour, en autorisant des sons stéréo. Le canal A sort sur le connecteur stéréo de

gauche, et le canal C sur celui de droite. Le canal B sort sur les deux, et tout ce qui en provient sera équilibré en volume à gauche et à droite. Des nombres supérieurs à 7 produisent des effets plus compliqués, que nous verrons plus tard.

Le deuxième nombre qui suit l'instruction SOUND est le « paramètre de hauteur ». Il contrôle la hauteur de la note produite par le CPC464, et sa plage de valeurs va de 1 (note la plus haute) à 4095 (note la plus basse). Il est plus pratique de limiter l'intervalle de 10 à 1000, parce que les notes au-delà de 1000 ressemblent plutôt à des séries de clics sauf si vous avez d'excellents hauts-parleurs. On peut utiliser 0, mais seulement pour des usages différents que nous verrons plus tard. Les notes que vous obtenez pour des nombres entre 1 et 10 sont trop hautes pour être efficaces. Les équivalents musicaux des nombres de hauteur sont clairement exposés dans le manuel, à l'*Appendice VII.* La page 2 de cet appendice montre l'ensemble le plus courant de notes, la gamme de do majeur. Le piano est l'instrument de musique le plus courant, et son clavier est conçu pour faciliter le jeu de cette gamme de do majeur. La gamme commence par une note appelée do majeur, et se termine sur la huitième note au-dessus, un do aussi, mais une octave au-dessus.

```
10 SOUND 1,478,50
20 FOR N=1 TO 2000:NEXT
30 SOUND 1,478,100
40 FOR N=1 TO 2000:NEXT
50 SOUND 1,478,200
```

Fig. 11.9. Une note unique simple.

Commençons notre investigation de l'instruction SOUND avec une simple note. C'est illustré à la *figure 11.9,* qui a sa première instruction SOUND à la ligne 10. Le canal A est choisi par le nombre 1 (premier paramètre) et la hauteur de la note par le nombre 478 (deuxième paramètre), ce qui donne approximativement un do majeur. Le ligne 10 fait entendre le son, et l'action de cette instruction SOUND fait que le son dure à peu près un cinquième de seconde.

Au point où nous en sommes, il est très important de com-

prendre ce qui se passe. Quand l'ordinateur exécute une instruction SOUND, il extrait les nombres, et les transmet à un système séparé, le générateur de sons. Cette partie de l'ordinateur travaille de son côté, et engendre le son physique. Entre-temps, cependant, l'ordinateur continue son rôle, et effectue l'instruction suivante. Ici aussi, deux choses peuvent être faites simultanément. Si vous avez beaucoup d'instructions SOUND les unes après les autres, l'ordinateur va les parcourir comme une flèche, en envoyant les paramètres au générateur de sons, puis continuer son propre travail. Chaque note sonne pendant un certain temps, et l'ordinateur peut avoir envoyé les données de cinq nouvelles notes au générateur de son avant que la première note ait fini de sonner! C'est ce que le manuel veut dire quand il parle de « liste d'attente sonore », et tant que vous n'aurez pas compris ce que cela veut dire, vous trouverez que les instructions SOUND plus compliquées ne produisent pas les effets que vous attendez d'elles.

Dans cet exemple, chaque SOUND est suivi d'une boucle d'attente assez longue pour que chaque note s'achève avant que l'ordinateur ne passe au SOUND suivant. La durée de chaque note est fixée par le nombre qui suit le paramètre de hauteur, le troisième. A la ligne 30, ce nombre est 50, et le temps est de 50/100e de seconde, ce qui fait une demi-seconde. Une valeur de zéro, ou des nombres négatifs ont des usages spéciaux, que nous verrons, eux aussi, plus tard.

Le test de la liste d'attente : SQ

La *figure 11.10* montre une variante au dernier programme, qui améliore l'espacement des notes. La nouveauté est ici SQ, qui renseigne sur la liste d'attente des sons, tout comme XPOS renseigne sur la position du curseur. Tant qu'il y a quelque chose dans la liste d'attente, SQ a une valeur au moins égale à 128. En testant cette valeur dans une boucle WHILE...WEND, on peut faire que l'ordinateur attende la liste, afin que l'instruction suivante ne soit pas ajoutée avant la fin de la première. On peut donc séparer ainsi les notes.

L'avantage est que, contrairement à une boucle FOR...NEXT, vous n'avez pas à avoir toujours le même intervalle de temps

```
10 SOUND 1,478
20 WHILE SQ(1)>128:WEND
30 FOR N=1 TO 100:NEXT
40 SOUND 1,478,50
50 WHILE SQ(1)>128:WEND
60 FOR N=1 TO 100:NEXT
70 SOUND 1,478,200
```

Fig. 11.10. Jouer des notes avec une pause fixe entre. SQ sert à trouver la fin d'une note.

entre deux instructions sonores. Une boucle FOR...NEXT met toujours le même temps à s'effectuer, une fois la valeur du compteur fixée, et la mesure commence dès l'instant que la note est mise sur la liste d'attente. Avec WHILE...WEND et SQ, vous mesurez le temps sur la note elle-même, et passez de la fin de la première au début de la seconde.

SQ doit être suivi du numéro de canal concerné entre parenthèses (il y a une liste d'attente par canal). D'autres usages de SQ, comme pour beaucoup d'instructions sonores, mériteraient un ouvrage à eux seuls !

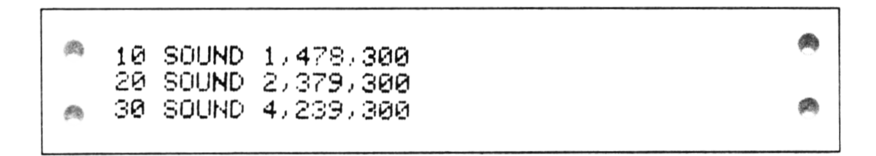
```
10 CLS:PRINT TAB(11)"GAMME DE DO MAJEUR"
:PRINT:PRINT
20 FOR N=1 TO 8
30 READ note
40 SOUND 2,note,100
50 NEXT
60 PRINT"Maintenant,toutes les notes son
t sur la file d'attente!"
100 DATA 478,426,379,358,319,284,253,239
```

Fig. 11.11. La gamme de DO majeur, interprétée par CPC464.

L'étape suivante consiste à essayer une gamme. La table des nombres et des fréquences donnée à l'*Appendice VII* du manuel du CPC464 est votre guide pour les notes musicales. En vous servant de cette table, vous arrivez au programme de la *figure 11.11*, qui joue la gamme de do majeur. Dans cette gamme les fréquences sont telles que le do une octave au-dessus du do majeur a exactement une fréquence double de ce dernier.

C'est également vrai des autres notes de la gamme, doubler la fréquence revient à monter la hauteur d'une octave. Diminuer de moitié la fréquence revient à descendre d'une octave. Les nombres de hauteur qui servent de paramètre à SOUND jouent dans le sens contraire, si bien qu'une montée d'une octave est effectuée en diminuant de moitié le nombre fourni au CPC464. Vous pouvez voir d'après la *figure 11.11* que le do majeur correspond à une valeur de 478, et que le do d'une octave au-dessus correspond à l'exacte moitié, 239. Cette exactitude n'est pas toujours possible, quand on utilise des nombres entiers, mais on reste autant que possible près de ces rapports.

La *figure 11.11* fait aussi comprendre le problème de la liste d'attente. Le générateur de sons du CPC464 peut accepter cinq notes sur une file d'attente. Parmi ces cinq, l'une se fait entendre et les quatre suivantes attendent dans la file. Quand on exécute le programme, donc, on voit le titre et on entend la gamme commencer. Quand cinq notes sont en attente, et qu'il y a plus de notes à lire, l'ordinateur doit lui aussi attendre. Mais quand toutes les notes restantes sont dans la file, l'ordinateur peut aller plus loin, et le message de la ligne 60 apparaît.



```
10 SOUND 1,478,300
20 SOUND 2,379,300
30 SOUND 4,239,300
```

Fig. 11.12. Un accord utilisant trois canaux.

L'étape suivante nous fait explorer l'usage de plus d'un canal sonore à la fois. La *figure 11.12* montre comment jouer un accord sur trois canaux. Nous n'avons pas encore regardé le contrôle du volume, et jusqu'ici le volume est le même sur les trois canaux, pour des raisons de simplicité. Mais pour obtenir de meilleurs effets musicaux, vous pouvez désirer utiliser des volumes différents sur différents canaux. Comme l'oreille humaine est moins sensible aux notes graves ou aiguës par rapport aux notes voisines de do majeur, il est souvent utile de donner plus de volume à ces notes qu'à celles du milieu de l'intervalle audible.

Pour qui a une bonne oreille, l'accord n'est pas tout à fait aussi juste qu'il devrait, parce que les nombres qui déterminent les hauteurs des notes sont seulement approximatifs. Si vous avez une oreille entraînée, vous pouvez faire des expériences de petites modifications dans les valeurs des paramètres de hauteur. Par exemple, je trouve qu'une valeur de 380 est plus acceptable (à la ligne 20) que 379.

Le quatrième nombre qu'on peut utiliser avec SOUND fixe le « volume relatif » de la note. On entend par volume relatif le fait que sans changer le réglage de volume de votre CPC464 (ou de votre amplificateur) vous pouvez modifier le volume du son (relativement au réglage de l'amplificateur) en modifiant la valeur du paramètre. Bien entendu on peut régler le volume absolu à l'aide du bouton de réglage de l'amplificateur ou du CPC464.

La plage de valeurs de ce nombre est, en nombres entiers, de 0 (silence) à 7 (plein volume). On peut utiliser des nombres de 0 à 15 quand le volume est contrôlé par une *enveloppe* — c'est encore un sujet à aborder. Si vous sortez de cette plage, vous aurez un message d'erreur "Improper argument" (argument impropre).

```
10 FOR N=0 TO 7
20 SOUND 2,339,50,N
30 NEXT
40 FOR N=7 TO 0 STEP -1
50 SOUND 2,339,50,N
60 NEXT
```

Fig. 11.13. Utilisation du paramètre contrôlant le volume.

La *figure 11.13* montre l'usage du paramètre de volume. Le programme commence par utiliser une boucle dans laquelle le nombre utilisé pour le volume augmente au cours de la boucle. Dans la deuxième partie, le paramètre de volume est diminué au cours de la boucle. L'effet est celui d'une note qui devient plus forte, avec des degrés sensibles, puis redevient plus douce.

Les notes produites par des instruments de musique traditionnels changent toujours ainsi de volume. Beaucoup d'ins-

truments produisent des notes qui commencent assez fort, et puis s'atténuent. Le piano est typique de ce genre, parce que la note est la plus forte au moment où la corde est frappée, puis s'atténue quand l'étouffoir est maintenu contre la corde. Le CPC464 vous permet d'imiter ce comportement (une « enveloppe ») d'autres manières, aussi n'est-il pas nécessaire d'utiliser des boucles pour contrôler l'amplitude.

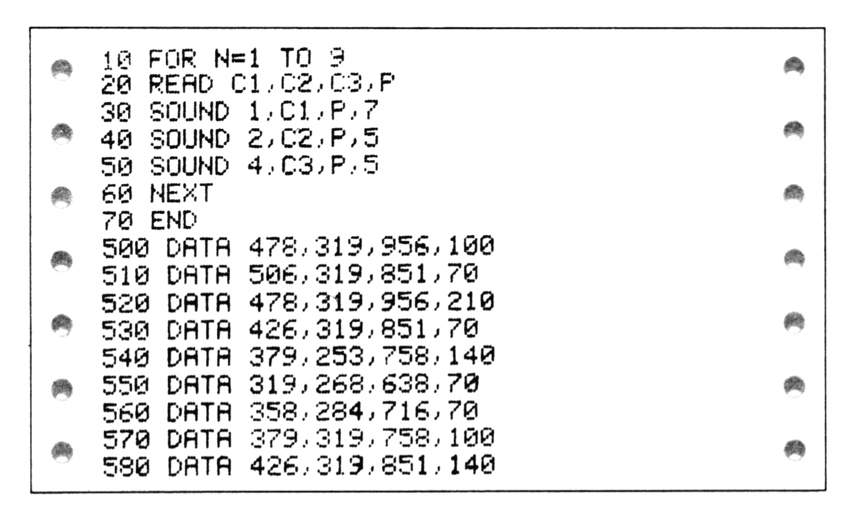
Musique, musique, musique

La capacité du CPC464 à produire des sons sur plus d'un canal vous permet de composer de la musique avec des accords, selon les règles de l'*harmonie.* A moins d'être un compositeur accompli, ou de vouloir l'être, il vaut mieux utiliser des partitions comme guide. La meilleure musique à utiliser, si vous voulez trois canaux, est la musique pour piano et violon, ou piano et voix de soprano, ou flûte et piano. Ces instruments donnent une tessiture de notes qu'un petit haut-parleur peut raisonnablement supporter. De la musique pour contre-basse ne sonne pas bien sur un petit haut-parleur ! Évitez la musique pour des instruments comme la clarinette ou le basson, parce que ce sont des instruments « à transposition » sur lesquels les notes jouées en fait ne sont pas les mêmes que celles que l'on entendrait sur un autre instrument pour la même musique.

La meilleure technique consiste à utiliser une boucle pour lire les données de hauteur et de durée de chaque note. Plus tard, nous verrons d'autres données à utiliser, mais mieux vaut commencer simplement. Pour vos premiers essais, il vaut mieux consacrer une ligne de DATA à chaque note. Si vous voulez laisser une note sonner sur un canal pendant que d'autres notes changent, vous devrez utiliser des méthodes de synchronisation que nous verrons dans un instant.

Vous constaterez que la musique pour piano sonne mieux si vous laissez de brèves pauses entre les notes, mais que la musique pour orgue n'en a pas besoin. Sitôt que vous aurez acquis les compétences de programmation, le principal sera pour vous de gagner de l'expérience.

Prenez comme exemple la *figure 11.14* qui illustre un morceau d'accords à trois voix. Cela n'a pas été repris d'une parti-



```
10 FOR N=1 TO 9
20 READ C1,C2,C3,P
30 SOUND 1,C1,P,7
40 SOUND 2,C2,P,5
50 SOUND 4,C3,P,5
60 NEXT
70 END
500 DATA 478,319,956,100
510 DATA 506,319,851,70
520 DATA 478,319,956,210
530 DATA 426,319,851,70
540 DATA 379,253,758,140
550 DATA 319,268,638,70
560 DATA 358,284,716,70
570 DATA 379,319,758,100
590 DATA 426,319,851,140
```

Fig. 11.14. Un peu d'harmonie, en utilisant trois canaux.

tion, et j'ai dû opérer en écrivant chaque canal séparément. C'est assez clair, parce que le programme principal consiste en une boucle, avec C1, C2 et C3 utilisés pour les notes des canaux A, B et C respectivement, et P pour la durée. Utiliser cette forme permet de modifier facilement l'air en changeant les lignes de DATA. J'ai écrit les nombres pour le canal A d'abord, en mettant des zéros pour les autres. Une fois les différentes valeurs ajustées, et les paramètres de durée mis en place pour un rythme raisonnable, j'ai obtenu l'air voulu, et je me suis mis au canal B, et après au canal C. Ce n'est pas aussi bien que cela l'aurait été avec une partition. A propos, c'est Beethoven l'auteur, je n'ai fait que la programmation.

Le département des effets spéciaux

L'instruction SOUND, même sous cette simple forme, peut produire une large gamme d'effets sonores. Commençons par une série de notes de hauteur ascendante, qui fait un avertisseur utile, ou la marque qu' « il va se passer quelque chose ». Cela est illustré à la *figure 11.15.* La boucle qui commence à la ligne 10 utilise des valeurs de J qui causeront la répétition de la note montante 10 fois. La boucle suivante, utilisant N, va de 200 à 1 par pas de —10. Elle aura pour effet de couvrir une bonne part de la plage audible, avec un rapide changement de

```
10 FOR J=1 TO 10
20 FOR N=200 TO 1 STEP -10
30 SOUND 2,N,1,7
40 NEXT
50 NEXT
```

Fig. 11.15. Programmation d'une note de hauteur croissante, en utilisant une boucle.

hauteur. Ces nombres sont ensuite utilisés comme paramètres de hauteur dans l'instruction SOUND de la ligne 30. On utilise une durée très courte, et le plein volume.

Essayez et écoutez soigneusement. Un des problèmes des instructions sonores est qu'il est impossible de décrire par écrit les résultats ! Contrairement à un programme graphique, pour lequel on peut se faire une idée du résultat en regardant le listing, un programme sonore est presque toujours une surprise, tant que vous n'avez pas une grande expérience. Il est donc important d'essayer chacun des programmes échantillons pour accroître votre expérience correctement. Ce serait une bonne idée de garder des enregistrements des sons produits que vous puissiez réentendre à l'aide d'un lecteur de cassettes ordinaire.

```
10 FOR J=0 TO 50
20 SOUND 2,239,5,7
30 SOUND 2,244,5,7
40 NEXT
```

Fig. 11.16. Un programme de note tremblée.

La *figure 11.16* montre un programme qui produit une note tremblée. C'est un procédé particulièrement utile pour attirer l'attention, ou pour annoncer un événement dans un jeu. Il semble qu'une note tremblée soit plus efficace qu'une note simple, d'où le choix de notes tremblées pour les sonneries des téléphones récents. Le trémolo de ce programme utilise la boucle qui commence à la ligne 10. Celle-ci fait entendre 50 paires de notes brèves d'une durée de 5 par note. Rappelez-vous que les durées sont en unité d'un centième de seconde.

Les deux paramètres de hauteur choisis dans cet exemple sont 239 et 244. Des hauteurs supérieures sont encore plus efficaces, et des valeurs comme 90 et 95 donnent des sonneries qui attirent vraiment l'attention.

Ajouter des bruits

Vous pouvez spécifier un autre type de son pour n'importe lequel des trois canaux, différent parce que vous obtenez non des notes musicales ordinaires, mais un *bruit*. Ce qu'on appelle un bruit est un mélange aléatoire de fréquences, mais on constate très souvent qu'une fréquence ou une plage de fréquences domine en volume les autres. Une source de bruits est utile parce qu'on peut l'utiliser pour produire des effets comme le son des vagues, un roulement de tambour, un coup de fusil, ou d'autres sons que l'on ne pourrait obtenir avec l'instruction SOUND telle que nous l'avons vue. Pour produire un bruit, on doit éliminer la source du son ordinaire en mettant à zéro le paramètre de hauteur. Le paramètre de « hauteur de bruit » peut prendre des valeurs de 0 à 15, et il doit former le septième paramètre de l'instruction SOUND. Je sais que nous avons sauté les paramètres 5 et 6, mais c'est exprès, et nous y reviendrons. Pour le moment, nous mettrons ces nombres à zéro.

```
10 CLS
20 FOR J=0 TO 15
30 PRINT "Période de bruit";J
40 SOUND 1,0,20,7,0,0,J
50 FOR N=1 TO 2000:NEXT
60 NEXT
```

Fig. 11.17. Production d'un bruit sur un canal.

La *figure 11.17* illustre un bruit sur le canal B. Il n'y a qu'une source de bruit, aussi ne pouvez-vous avoir différents bruits sur chacun des trois canaux. Mais vous pouvez évidemment placer le bruit sur n'importe lequel des trois. Avec un amplificateur stéréo et les haut-parleurs face à vous, par exemple, vous pouvez vous donner les sensations d'OK Corral, avec des coups de feu à droite et à gauche !

```
10 CLS:PRINT:PRINT"Les vagues sur le riv
age..."
20 FOR N=1 TO 10
30 FOR J=15 TO 1 STEP -1
40 SOUND 2,0,20,7,0,0,J
50 NEXT:NEXT
60 PRINT:PRINT"Le marteau du chaudronnie
r..."
70 FOR N=1 TO 10
80 FOR J=15 TO 0 STEP -1
90 SOUND 2,0,5,J,0,0,1
100 NEXT:NEXT
```

Fig. 11.18. Comment utiliser le générateur de bruits pour produire des effets sonores.

Le nombre utilisé pour la « période du bruit » n'a pas le même effet, ni la même plage que pour les autres canaux. La plage va de 0 à 15, et les effets se répètent simplement si vous utilisez des nombres plus grands. Vous constaterez que tous les nombres sont utiles pour des effets, mais qu'il n'y a guère de différence entre les effets produits par les nombres de 1 à 5. Pour mieux apprécier le canal de bruit, essayez le programme de la *figure 11.18.* Il donne deux splendides effets, grâce à la programmation d'un bruit dans une boucle, en variant différentes choses à chaque passage. Dans le premier ensemble, dix bruits de « vagues » sont produits. C'est fait en changeant la période du bruit dans la boucle qui utilise la variable J. Le bruit commence par une prédominance de basses fréquences, qui passe peu à peu à de plus hautes fréquences. La deuxième partie du programme produit des bruits de martèlement. Dans ce cas, le volume est modifié dans la boucle, et la période du bruit est fixe. La petite période du bruit produit une note de martèlement métallique ; de plus grands nombres donnent des hauteurs plus basses, qui sonnent très différemment.

Des formes d'ondes à l'infini : les enveloppes

Il est temps maintenant d'explorer l'effet de l'extension de SOUND, pour contrôler les « enveloppes » du son que vous entendez. L'effet n'a rien de simple, et comme c'est difficile

de décrire verbalement ce à quoi ressemble un bruit, vous devrez essayer les programmes et écouter ! Tout d'abord, je dois expliquer ce qu'on entend par « enveloppe ». Les sons que la simple instruction SOUND produit ont une amplitude et une fréquence constantes. Pour parler plus simplement, leur force et leur hauteur restent constantes pendant la durée de la note. Mais les instruments de musique produisent des notes dans lesquelles la force varie au cours même de l'émission de la note. Une note de piano, par exemple, est forte au moment où elle est jouée, parce que c'est là que la corde est frappée. Ensuite, le son s'amortit rapidement, et c'est la manière dont le son s'amortit qui rend les notes de piano si reconnaissables.

D'autres instruments produisent des notes qui se comportent très différemment, et tous produisent en fait des notes qui consistent en un mélange de fréquences. C'est ce qui vous permet de distinguer un piano d'un violon, ou une flûte, ou un hautbois, même si tous jouent la même note. La courbe d'amplitude d'une note, tracée suivant le temps de son émission, est appelée l'enveloppe de volume de cette note.

De plus, la hauteur de la note n'est pas constante non plus. Si vous avez jamais vu un violoniste en train de jouer, vous aurez remarqué comment la main qui presse la corde est secouée de part et d'autre. Cela donne un effet de trémolo, de changement rapide de hauteur. Ajouter un trémolo à une note peut rendre le son plus intéressant que celui d'une note de hauteur constante. Une note qui a du trémolo, ou toute autre variation de hauteur, aura une enveloppe de hauteur.

Regardez la *figure 11.19.* Elle montre une enveloppe de volume typique de beaucoup de notes de musique. Elle présente une amplitude croissant rapidement au début, que nous appelons l'*attaque*. Cette phase est suivie du *déclin*, phase où l'amplitude s'affaiblit. Puis l'amplitude reste stable un moment, dans la phase de *tenue*, et ensuite tombe à zéro dans la section de *relâchement*. Le CPC464 nous donne l'occasion de synthétiser une pareille enveloppe en utilisant une version étendue de l'instruction SOUND. Le principe est d'utiliser une instruction qui peut créer différentes formes d'enveloppes. Chacune de ces enveloppes reçoit un numéro, et l'on peut utiliser ce numéro dans une instruction SOUND — c'est le cinquième paramètre de SOUND. Quand on utilise

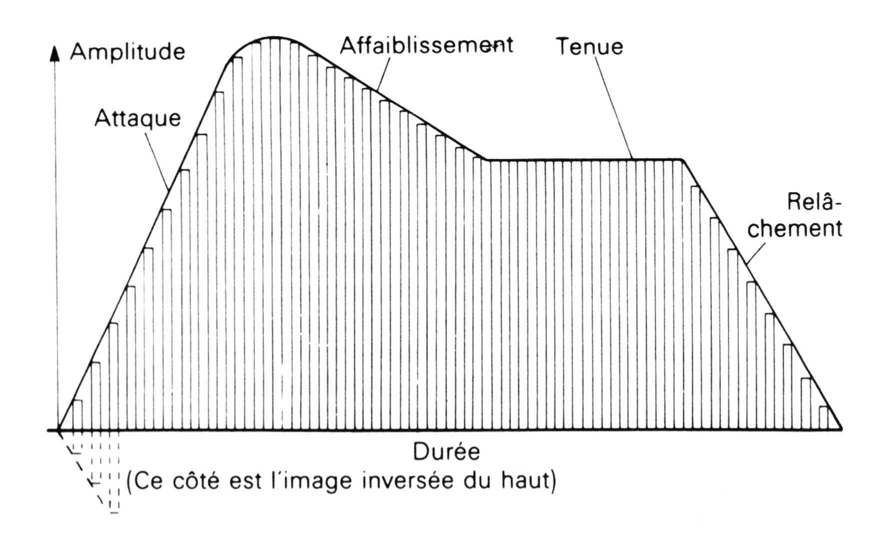


Fig. 11.19. Une « enveloppe » typique de volume sonore, qui montre comment le volume d'une note évolue pendant que vous l'entendez.

une enveloppe de volume, il faut lui laisser contrôler à la fois le volume et la durée de la note. Si on utilise zéro pour le paramètre de durée (le troisième nombre), l'enveloppe prendra le contrôle. Le nombre de volume utilisé (le quatrième paramètre) décide du volume de la note au début de son émission, ensuite l'enveloppe prend effet.

La fermeture de l'enveloppe

Pour créer une enveloppe de volume, on doit utiliser l'instruction ENV, qui est suivie d'au moins 4 nombres, et au plus de 16. A première vue, cela paraît un peu effrayant, mais comme tous les aspects de ce remarquable système sonore, cela vaut la peine de le maîtriser. Voyons comment faire, avec des exemples simples.

Pour commencer, nous avons besoin d'une grille de préparation, comme nous en montre une la *figure 11.20.* Les pas de volume utilisent des degrés de 0 à 15, qui constituent la plage

admise par cette instruction, pour le volume. La plage de durée telle qu'elle est montrée va de 0 à 50. Avec des unités de temps d'un centième de seconde, cela donne des enveloppes qui vont jusqu'à une demi-seconde. Pour des enveloppes plus longues, vous pouvez tracer votre propre courbe, en utilisant une échelle différente. Le principe est de tracer votre repère, de faire votre courbe dessus, et d'essayer de suivre la courbe d'aussi près que possible, en respectant les pas de volume. Vous ne pouvez pas avoir de valeurs de volume décimales, aussi toute partie droite de la courbe, si elle est en pente, doit être représentée comme une série de marches suivant le pas. La seule chose que vous puissiez faire est de décider combien de pas vous allez utiliser, — pas plus de 15, puisque c'est le nombre de pas de volume.

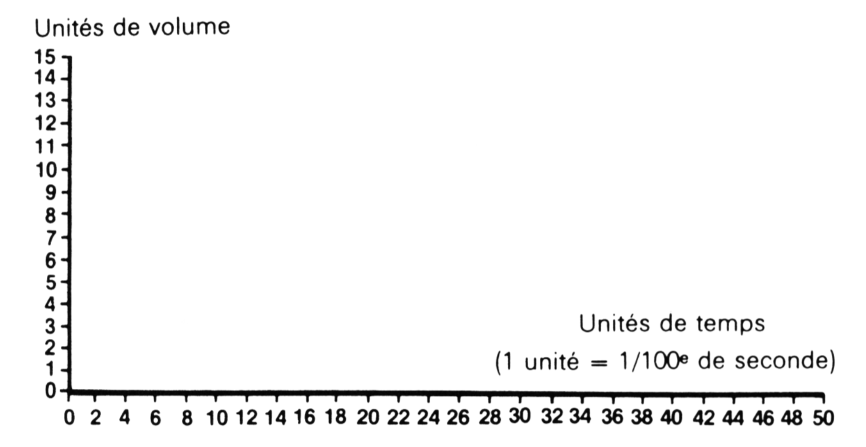


Fig. 11.20. La grille de préparation pour l'instruction ENV. Vous pouvez tracer vous-même ces grilles, sur du papier millimétré.

Prenons l'exemple le plus simple possible, celui d'un son qui passe du volume maximal à zéro, selon une diminution constante, en ligne droite, comme le montre le trait épais de la *figure 11.21*. Maintenant, voyons combien de pas nous pouvons utiliser. Pour garder sa simplicité à l'exemple, le plan montre cinq étapes, chacune diminuant le volume de trois unités, et chacune durant quatre unités de temps. Le temps qui est mesuré là est le temps passé sur la partie horizontale de chaque pas — ce que le manuel appelle le temps de « pause ».

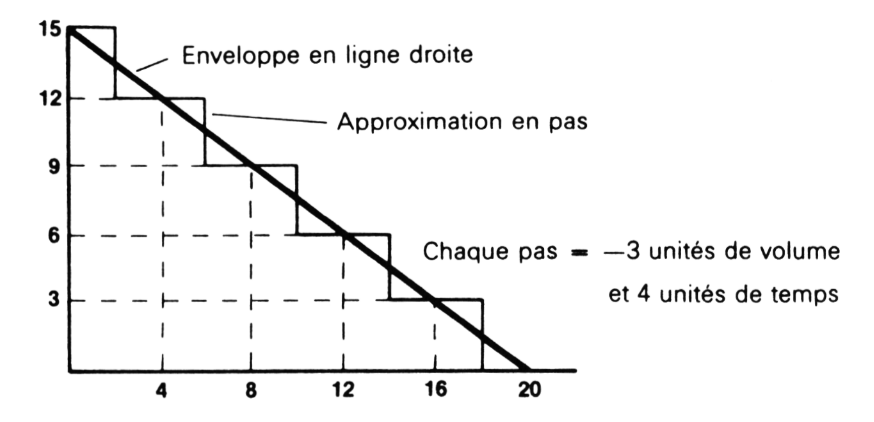


Fig. 11.21. Une forme d'enveloppe simple, à un seul pas.

Donnons à cette enveloppe le numéro 1. Maintenant, il faut programmer cela avec ENV, suivi du numéro, qui est 1, puis du nombre de pas, de la taille du pas, et du pas de temps. Cela nous donne ENV 1,5,-3,4. Nous pouvons introduire cela dans un programme et écouter le résultat (voir *figure 11.22*).

```
10 CLS
20 ENV 1,5,-3,4
30 PRINT "La durée du zéro"
40 SOUND 2,239,0,15,1,0,0
50 FOR N=1 TO 2000:NEXT
60 PRINT"La durée négative= 10"
70 SOUND 2,239,-10,15,1,0,0
```

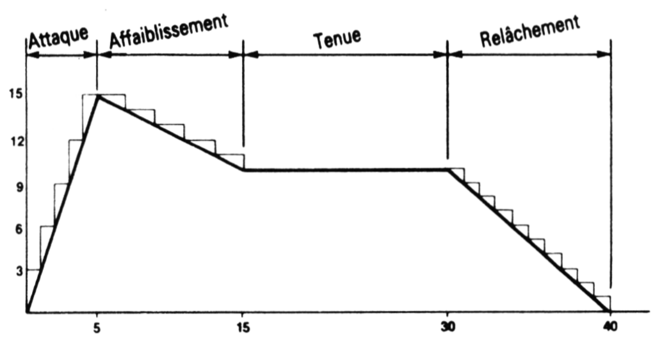
Fig. 11.22. L'enveloppe « ouverte » — écoutez !

Ce programme spécifie notre enveloppe simple, ENV 1,5,-3,4, et ensuite SOUND utilise cette enveloppe. Le canal sélectionné est 2, et la fréquence 239. La durée est 0, ce qui permet à l'enveloppe de choisir la durée. Notre conception de l'enveloppe a autorisé une durée totale de 18 unités, ce qui fait une durée de 18/50 de seconde. Le volume de départ est 15, le maximum. Quand on utilise une enveloppe, on a droit à un maximum de 15 et un minimum de 0. Après le paramètre de volume, qui règle le début du son, on place le numéro d'enveloppe de volume, c'est-à-dire 1. L'enveloppe de hauteur est 0,

puisque nous ne l'avons pas encore utilisée, et la période de bruit est à zéro puisqu'il ne s'agit pas d'un bruit. Tout cela est programmé à la ligne 40, et vous pouvez entendre l'effet à l'exécution du programme. La ligne 70 illustre l'effet de l'usage d'un nombre négatif comme paramètre de durée : cela fait répéter l'enveloppe pour le nombre spécifié de fois, sans tenir compte du signe —. Un paramètre de durée à —10, par exemple, fait répéter l'enveloppe dix fois.

Rien qu'avec cette enveloppe très simple, qui n'utilise que quatre nombres avec ENV, il y a beaucoup de perspectives expérimentales. Pour ne faire qu'entamer un riche filon, mettez le paramètre de hauteur à 1, et prenez un bruit, disons 12. Cela donne un bruit de brusque coup de feu, et il y a beaucoup de variations possibles autour de ce thème, avec des mélanges de note et de bruit.

Pour plus d'effet cependant, vous pouvez avoir *jusqu'à cinq sections* d'enveloppe, et vous pouvez utiliser des pas bien plus fins que ceux de l'exemple. Il est mieux d'utiliser autant de pas que permis à chaque stade par la durée, avec un maximum de 15. Essayons une enveloppe plus ambitieuse, qui contient quatre sections, attaque, déclin, tenue et relâchement.



Attaque : 0 à 15 en 5 unités de temps	**Tenue**
Pas = 5	Pas = 1
Taille = 3	Taille = 0 (pas de changement)
Temps = 1	Temps = 5
Affaiblissement : 15 à 10 en 10 unités de temps	**Relâchement** : 10 à 0 en 10 unités de temps
Pas = 5	Pas = 10
Taille = —1	Taille = —1
Temps = 2	Temps = 1

Fig. 11.23. Plan d'une enveloppe à quatre sections.

C'est illustré sur la *figure 11.23*. Une fois de plus, la ligne épaisse montre l'objectif visé, et les marches montrent l'approximation. Il faudra une instruction ENV avec quatre sections, et les nombres à utiliser dans chaque section apparaissent sur la droite du diagramme. Essayons, à la *figure 11.24*.

```
10 ENV 1,5,3,1,5,-1,2,1,0,15,10,-1,1
20 SOUND 2,478,-10,0,1,0,0
30 FOR N=1 TO 2000:NEXT
40 FOR J=1 TO 8
50 READ note%
60 SOUND 2,note%,0,0,1,0,0
70 NEXT
100 DATA 478,379,426,638,638,426,379,478
```

Fig. 11.24. Le programme SOUND qui utilise cette enveloppe.

L'instruction ENV commence par spécifier le numéro d'enveloppe, puis énumère le nombre de pas de volume, la taille des pas, et la durée du pas pour chaque section. Une fois cela fait, on crée le son, en utilisant une valeur de durée de —10 pour donner dix coups d'une hauteur. Aux lignes 40 à 70, on essaye un carillon avec cette enveloppe. Un peu vite, n'est-ce pas ? C'est facile à corriger, ajoutez simplement un cinquième stade de silence à l'enveloppe. Ajoutez les nombres 1,0,40 à la fin de l'instruction ENV, et écoutez la différence !

Travailler avec ces enveloppes demande un peu de préparation, une bonne oreille, et beaucoup d'entraînement. Bien que les règles d'usage des paramètres d'enveloppe soient en fait très directes, et bien plus simples que les méthodes adoptées sur d'autres machines, vous pouvez vous simplifier la vie en utilisant quelques astuces. Essayez de donner à vos pas une taille raisonnable. Ayez un nombre raisonnable de pas. Si vous faites une enveloppe trop courte, il n'y aura pas grand-chose à entendre, sauf si vous créez des bruits comme des coups de pistolet. En tout cas, pour les notes de musiques, les enveloppes de courte durée se révéleront très décevantes. En règle générale, visez des notes qui durent plus d'une seconde. Une autre astuce consiste à essayer de tracer des changements d'amplitude sur les diagonales des carrés de la grille.

Cela vous assure que vous aurez un nombre entier de pas, ainsi qu'une taille entière pour ceux-ci.

Les trémolos

Nous avons vu plus tôt que l'on pouvait aussi utiliser une enveloppe de hauteur pour une note. L'enveloppe de hauteur est programmée de manière analogue à celle d'amplitude, et une grille de préparation est illustrée à la *figure 11.25*. Là encore, la forme voulue est approchée en marches d'escalier. Mais cette fois, les pas sont des valeurs de hauteur. Un changement positif de hauteur signifie une baisse de hauteur, un pas négatif signifie une montée. Comme d'habitude un exemple sera utile, et la *figure 11.26* en donne un simple. Cette enveloppe fera monter la note pour les premiers 8/100e de seconde, puis redescendre pour les 8/100e suivants. Comme précédemment, tracer les lignes en suivant les diagonales des carrés rend la préparation plus simple à réaliser. La *figure 11.27* illustre le type de son que l'on obtient. C'est bon pour les cloches, les ressorts et autres bruits bizarres !

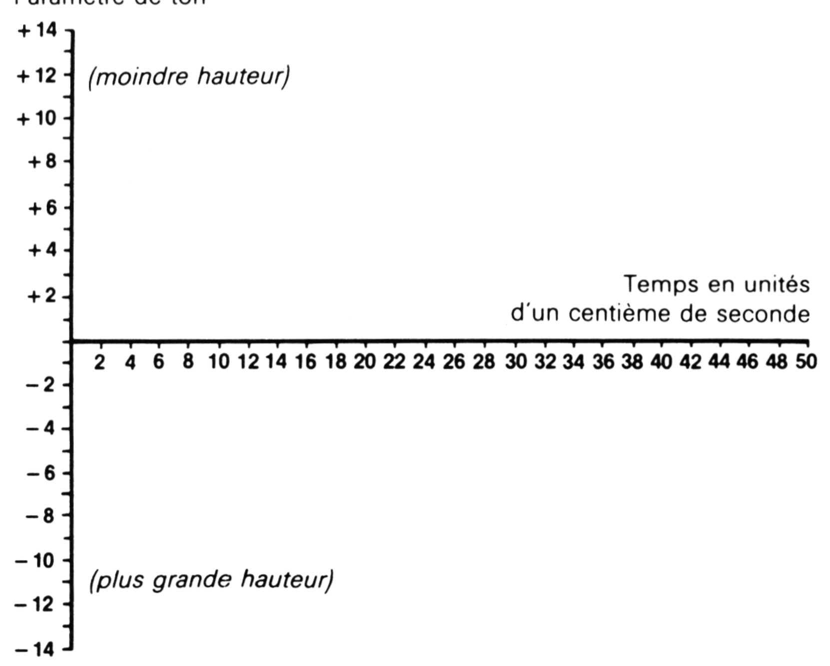


Fig. 11.25. La grille de préparation d'une enveloppe de hauteur.

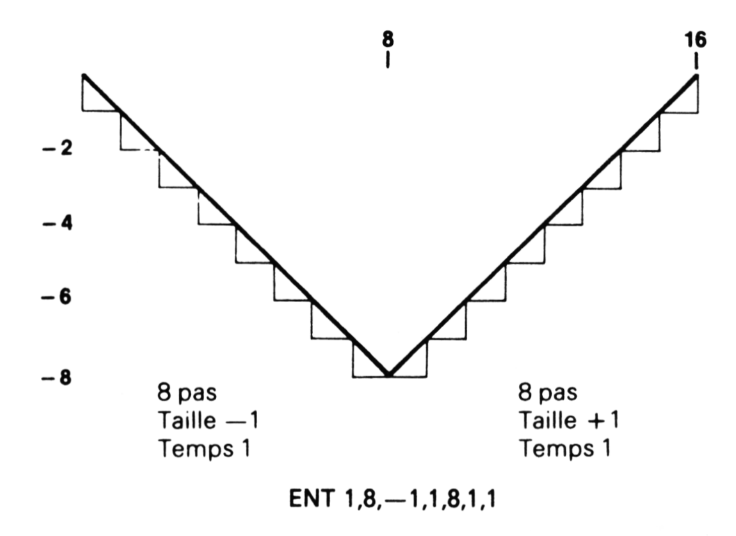


Fig. 11.26. Un plan simple d'enveloppe de hauteur.

```
10 ENT 1,8,-1,1,8,1,1
20 SOUND 2,478,50,7,0,1,0
30 FOR N=1 TO 2000:NEXT
40 FOR J=1 TO 8
50 READ note%
60 SOUND 2,note%,40,7,0,1,0
70 NEXT
100 DATA 478,379,426,638,638,426,379,478
```

Fig. 11.27. Comment sonne cette enveloppe de hauteur.

Pour les notes de musique, un effet de trémolo est obtenu en haussant la fréquence, puis en la baissant selon le schéma de la *figure 11.28.* C'est un trémolo un peu lent, et dans beaucoup de cas, il faudrait une variation plus rapide. Dans le programme de la *figure 11.29* le numéro d'enveloppe de hauteur présente une particularité : le numéro est —1, bien que dans l'instruction SOUND l'enveloppe soit désignée par 1. L'effet du signe — dans l'instruction ENT est de faire répéter l'enveloppe. Si on avait utilisé 1, le trémolo se serait produit une fois au début de la note, alors qu'avec —1 le trémolo est répété tant que la note dure.

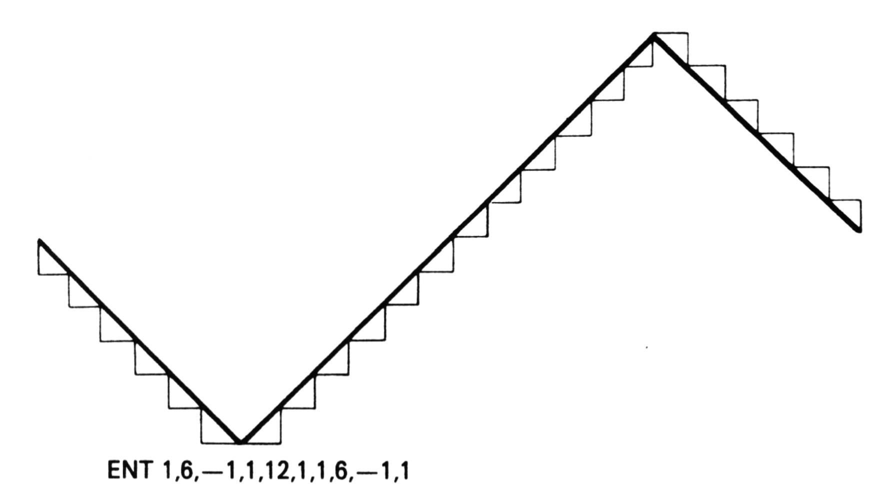


Fig. 11.28. Un tracé de trémolo pour une enveloppe de hauteur.

```
10 ENT -1,6,-1,1,12,1,1,6,-1,1
20 SOUND 2,478,200,7,0,1,0
```

Fig. 11.29. Utilisation de l'enveloppe de trémolo. Notez l'usage de ENT —1 pour faire répéter l'enveloppe.

Les combinaisons de bruits avec les enveloppes de hauteur sont aussi fascinantes que celles avec les enveloppes de volume. Pour un joli exemple, prenez le SOUND de la *figure 11.29,* mettez le paramètre de hauteur à 1, et celui de bruit à 7, changez la durée en 1200. Exécutez ce programme, et écoutez le son d'une locomotive à vapeur peinant dans une côte !

Il est inutile de préciser que l'on peut combiner enveloppe de volume et enveloppe de hauteur, pour faire à peu près n'importe quel son à votre gré. La démarche à suivre est d'explorer systématiquement les possibilités sonores. Enregistrez toujours les sons obtenus, et gardez trace des enveloppes utilisées. Ainsi vous pouvez vous construire une bibliothèque d'effets sonores que vous trouverez extrêmement utile pour vos programmes, ou pour ceux des autres. N'oubliez pas non plus que les revues sont avides de programmes, et paient les auteurs à la page de listing.

Synchronisation

Faute d'espace on traitera brièvement de ce sujet. Faire justice au générateur de son du CPC464 exigerait un autre livre entier! On omettra donc le détail de l'instruction SQ. Dans cette dernière partie, je voudrais jeter un coup d'œil à la *synchronisation.*

Quand vous empilez un ensemble de notes sur la file d'attente de chaque canal, il devient plus difficile de vous assurer qu'elles sont jouées correctement. En fait, c'est presque impossible sur certains ordinateurs. En particulier, vous pouvez désirer faire sonner deux notes ensemble en même temps, puis faire taire un canal pendant qu'une autre note sonne sur le premier canal, et ainsi de suite.

Choix n°	Effet
8	Synchronisation avec canal A
16	Synchronisation avec canal B
32	Synchronisation avec canal C
64	Attente
128	Nettoyage des tampons des listes d'attente

Fig. 11.30. La plage entière de sélection des canaux avec synchronisation.

Le CPC464 utilise un système de numéros de synchronisation pour obtenir ce résultat. La synchronisation est obtenue en utilisant les numéros de choix de canaux, et la *figure 11.30* montre la gamme complète de valeurs, avec leur effet. Tout comme les nombres qui envoient les signaux sonores sur les trois canaux, il y a trois nombres qui causent la synchronisation, un qui fait attendre une note dans la file, et un qui enlève tout ce qui peut se trouver dans les files — ce dernier est un moyen rapide d'obtenir le silence. On peut obtenir des effets composites en additionnant les nombres.

La *figure 11.31* illustre ce principe. La ligne 10 est une instruction SOUND. D'ordinaire, on entendrait dès son exécution le son, mais en choisissant le canal 33, ce n'est pas le cas. Ce choix de canal est la combinaison de 1, qui ouvre normalement le canal A, et de 32 qui demande une synchronisation

```
10 SOUND 33,478,300
20 FOR N=1 TO 2000:NEXT
30 SOUND 12,379,300
```

Fig. 11.31. Une brève illustration de la synchronisation.

avec le canal C. Le résultat est que le son est maintenu dans la file d'attente, et que vous n'entendez rien jusqu'à ce que le canal C sonne. Après la pause, on arrive à l'instruction SOUND pour le canal C. Celle-ci a pour numéro de canal 12. C'est l'addition de 4, pour le canal C, plus 8 pour la synchronisation avec le canal A. Notez que *les deux paramètres de canal doivent contenir une valeur de synchronisation* pour en obtenir une. L'effet, quand vous exécutez ce programme, est qu'on n'entend rien jusqu'à la fin de la pause, puis l'accord se fait entendre.

Si tout cela semble un peu intimidant, souvenez-vous que, pour bien des effets, vous n'en avez pas besoin. Vous pouvez très aisément lire un ensemble de nombres de choix de canal et de hauteurs dans une liste de DATA, en utilisant la hauteur 0 pour le silence. A moins d'être réellement fort en composition musicale, vous pouvez négliger les problèmes de synchronisation ! Néanmoins, jouir de cette possibilité sur un ordinateur de ce prix est remarquable, et comme tous les talents, cette qualité ne disparaît pas parce que vous ne l'utilisez pas encore.

Appendice A

Édition de ligne

Éditer signifie changer quelque chose qui est déjà apparu sur l'écran. Vous pouvez évidemment effacer un caractère en utilisant la touche DEL, ou retaper complètement une ligne erronée. Vous pouvez aussi détruire une série de lignes en utilisant l'instruction DELETE (comme DELETE 10-100). Mais éditer signifie changer un élément d'une ligne sans avoir à changer tout le reste. N'importe quel élément de la ligne — y compris le numéro de ligne — peut être changé à l'édition. On peut effectuer une édition.

a) pendant qu'on introduit une ligne, avant d'avoir appuyé sur ENTER ;
b) après avoir appuyé sur ENTER pour enregistrer la ligne, et avant d'avoir exécuté le programme ;
c) quand une erreur est signalée en cours d'exécution.

En prenant les choses dans cet ordre :
a) pendant qu'on introduit une ligne, toutes les instructions d'édition de ligne ci-dessous peuvent être utilisées. L'édition est achevée lorsque l'on appuie sur ENTER.
b) quand la ligne a été validée mais que le programme n'a pas encore été exécuté, vous pouvez utiliser indifféremment l'une ou l'autre des méthodes détaillée ci-dessous.
c) quand le programme s'arrête avec un message d'erreur, la ligne sur laquelle l'erreur a été détectée peut apparaître à l'écran, avec le curseur sur le numéro de ligne. Vous pouvez éditer l'erreur en utilisant la méthode d'édition de ligne.

Méthodes d'édition

C'est un trait unique au CPC464 que de permettre deux méthodes d'édition. La plupart des ordinateurs utilisent soit

l'édition de ligne soit l'édition d'écran. L'édition de ligne signifie que vous devez appeler la ligne à modifier, en utilisant une instruction comme EDIT 200 (pour éditer la ligne 200). Le curseur peut alors être déplacé sur la ligne pour localiser et corriger l'erreur.

En édition d'écran, toute ligne visible sur l'écran peut être éditée simplement en amenant le curseur dessus, et en corrigeant l'erreur. La plupart des gens sont des partisans ou des ennemis farouches de telle ou telle méthode, et n'en utiliseront qu'une. En offrant les deux, la CPC464 satisfait tout le monde !

Instructions d'édition

1. Édition de ligne

Si vous n'avez pas encore la ligne sur l'écran avec le curseur dessus (comme quand une erreur est signalée), utilisez l'instruction EDIT pour mettre la ligne en place. Souvenez-vous qu'il doit y avoir un espace entre le T de EDIT et le premier chiffre du numéro de ligne. Le résultat sera de placer la ligne sur l'écran avec le curseur sur le premier chiffre du numéro de ligne. Vous pouvez alors déplacer le curseur sur la ligne en utilisant les flèches. Vous pouvez détruire un caractère sous le curseur en appuyant sur la touche CLR. En maintenant cette touche appuyée vous pouvez détruire tout ce qui se trouve à la droite du curseur. Inversement, vous pouvez détruire tout ce qui se trouve *à gauche* du curseur en utilisant la touche DEL. Taper un caractère l'insérera *à la position du curseur.* Appuyez sur ENTER pour achever l'édition de la ligne. Souvenez-vous que vous pouvez éditer une ligne ainsi pendant que vous la tapez. Si vous avez tapé.

PRINT C'EST LA FIN »

et que vous vous aperceviez soudain que vous avez oublié d'ouvrir les guillemets, utilisez la flèche vers la gauche pour amener le curseur sur le C, appuyez sur les guillemets puis sur ENTER. Il est important de retenir à propos de ce type d'édition de ligne que *vous pouvez appuyer sur ENTER dès que vous avez corrigé la faute*; vous n'avez pas à déplacer le curseur jusqu'à la fin de la ligne.

2. Édition par copie

L'édition par copie est la version propre au CPC464 de l'édition d'écran. Cette méthode fait une copie de ligne, mais vous permet d'omettre ou de remplacer des éléments de la ligne copiée. Tout ce que vous voyez sur l'écran, même les commandes directes, peut être copié.

Appuyez sur SHIFT et maintenez cette touche enfoncée. Maintenant déplacez le curseur à l'aide des flèches jusqu'à la ligne que vous voulez éditer. Si vous relâchez maintenant les touches du curseur (en maintenant SHIFT enfoncée) et appuyez sur la touche COPY, vous faites une copie de la ligne au bas de l'écran. La copie commence à l'emplacement du curseur. Si vous voulez sauter une partie de la ligne, utilisez la touche du curseur au lieu de COPY. Appuyez sur ENTER quand vous avez copié ce que vous voulez. Si vous voulez copier tout le reste de la ligne, *vous devez achever la copie avant d'appuyer sur ENTER.* Si vous appuyez sur ENTER au milieu d'une ligne, seule la partie déjà copiée sera enregistrée. C'est une différence importante entre l'édition de ligne et l'édition par copie.

Cette méthode est très souple. Vous pouvez, par exemple, changer un numéro de ligne. Supposez que vous ayez une ligne 200 sur l'écran et que vous vouliez en faire la ligne 1000. Tapez 1000, utilisez SHIFT et les flèches pour mettre le curseur sur la première instruction de la ligne 200, après le numéro de ligne. Maintenant copiez la ligne avec la touche COPY. Appuyez sur ENTER, et vous avez une ligne 1000 identique à la ligne 200. Tapez 200 et ENTER pour supprimer la ligne 200, et vous restez avec la ligne 1000.

Vous pouvez même copier quelque chose qui n'avait pas de numéro de ligne. Supposez qu'au milieu de l'introduction d'un programme, vous tapiez FOR N=1 TO 200:NEXT en oubliant le numéro de ligne. Vous n'avez pas à tout retaper. Contentez-vous de taper le numéro de ligne correct, puis utilisez les flèches pour placer le curseur sur le F de FOR et copiez la ligne. Appuyez sur ENTER à la fin de la ligne, et c'est en place !

Un autre trait avantageux de cette méthode, qu'on utilise aussi sur le micro BBC, est que l'on peut introduire un morceau d'une ligne dans une autre (ce qui est impossible avec

l'édition de ligne). Si vous décidez que la pause utilisée en ligne 400 est aussi utile dans la ligne 700, c'est simple. Mettez les deux lignes sur l'écran. Copiez la ligne 700 jusqu'à l'endroit où vous voulez la pause. Déplacez le curseur jusqu'à la ligne 400 où se trouve la routine, et copiez celle-ci. Revenez ensuite copier le reste de la ligne 700 si nécessaire. Appuyez sur ENTER quand c'est fini.

Élimination des erreurs

En jargon d'informatique, une erreur dans un programme est appelée une *bogue* (ou selon le terme anglais un *bug*), et la personne qui l'a écrite s'appelle bien sûr un programmateur. Vos programmes peuvent présenter toutes sortes de bogues, qui sont indiquées par les messages d'erreurs que vous obtenez à l'exécution. Certains de ces messages sont très explicites. « Line does not exist » (La ligne n'existe pas), par exemple, se rencontre quand vous avez utilisé une instruction comme GOTO 1000 ou GOSUB 1000 alors que la ligne 1000 n'est pas écrite. Cela peut aussi arriver si vous essayez de détruire la ligne 1000 par DELETE 1000 alors que celle-ci n'existe pas, ou si vous avez un THEN 1000 ELSE 2000 après un IF quelque part.

Le message d'erreur le plus fréquent est « Syntax error : (Erreur de syntaxe). Celui-ci signifie que vous avez mal utilisé un des mots réservés du BASIC. Vous pouvez avoir fait une faute d'orthographe, comme PRIBT au lieu de PRINT. Vous pouvez avoir oublié une parenthèse, une virgule, un point-virgule, ou mis un point-virgule à la place de deux points. La machine ne peut pas vous dire quelle est votre intention ; elle ne peut qu'exécuter servilement ce que vous lui dites. Si vous n'avez pas utilisé le BASIC exactement comme elle l'attend, vous constaterez qu'elle vous signale une erreur de syntaxe.

Une autre erreur commune est « Improper argument » (Argument impropre). Cela signifie d'ordinaire qu'il s'est passé quelque chose d'anormal avec un nombre. Vous avez pu, par exemple, utiliser TAB(300). Évidemment, après avoir lu ce livre, vous n'écririez pas TAB(300). Mais vous pourriez avoir TAB(N), et si N est arrivé à valoir 300, vous aurez le message d'erreur. Pour tout ce qui utilise des nombres, comme MID$, LEFT$, RIGHT$, INSTR, STRING$ entre autres, on peut ren-

contrer un argument numérique erroné. Vous pouvez aussi constater qu'utiliser un argument négatif dans SQR(N), un argument négatif ou nul dans LOG(N), et autres impossibilités mathématiques suscite le message « Improper argument ». La cause ne devrait pas être difficile à trouver, étant donné que la machine spécifie le numéro de ligne du problème.

Beaucoup d'erreurs peuvent s'introduire dans un programme, même quand vous recopiez un programme imprimé dans une revue. En général, les programmes publiés dans les mensuels d'informatique sont assez sûrs, mais il arrive qu'ils soient imprimés de manière difficilement lisible. Les principaux problèmes arrivent quand l'auteur du programme a utilisé I (I majuscule) ou l (L minuscule) comme nom de variable, ou a utilisé une imprimante qui n'a pas de zéro barré. La pire des confusions est celle entre O et 0. Une ligne comme IFM=10ORJ=4ORD=2 peut causer bien des ennuis, et il faudrait imprimer IF M=10 OR J=4 OR D=2 pour que tout soit clair. Quelquefois, omettre les espaces est nécessaire pour faire tenir le programme en mémoire, mais c'est la seule excuse réellement acceptable.

Même quand vous avez éliminé toutes les erreurs de syntaxe, et d'argument impropre, vous pouvez trouver que votre programme ne fait pas ce que vous attendez de lui. Le CPC464 fait ce que toute machine des années 80 devrait faire, il vous donne beaucoup de manières de trouver exactement ce qui ne va pas. L'un des moyens les plus puissants est l'usage de la touche ESC. Comme vous le savez, elle interrompt l'action de la machine quand vous l'enfoncez une fois, et le programme repart quand vous appuyez sur une autre touche. Comme nous le verrons plus tard, c'est très utile pour débusquer les erreurs dans un programme graphique, mais pour d'autres programmes, appuyer sur ESC deux fois est souvent plus utile. Cela arrête le programme, et affiche le numéro de ligne où s'est arrêtée l'exécution. Ce que vous ne savez probablement pas, c'est que vous pouvez afficher les valeurs des variables du programme, et même les modifier pendant l'arrêt du programme, et que vous pouvez reprendre l'exécution en utilisant un GOTO.

Supposez par exemple que vous exécutiez un programme qui utilise un comptage lent, que vous appuyiez deux fois sur ESC

assez tôt. Le programme s'arrête et vous avez un message comme « Break in 40 ». Ce numéro de ligne, 40 est important parce que vous pouvez relancer le programme à partir de cette ligne, pourvu que vous n'éditiez, ne détruisiez ni n'ajoutiez de ligne au programme. Supposez que le compteur soit N. Essayez de taper ?N, et ENTER. Cela vous donnera la valeur de N. Essayez maintenant N = 998 (disons) et ENTER. Tapez GOTO 40 (ou le numéro de ligne auquel le programme s'est arrêté). Vous verrez le compte reprendre, mais à 998 ! C'est une excellente manière de tester ce qui se passe à la fin d'une longue boucle. Tester toute la boucle prendrait beaucoup de temps si vous deviez laisser le compteur aller jusqu'au bout.

Vous pouvez même rendre ce processus automatique ! Nous avons déjà vu l'instruction ON BREAK GOSUB. Elle vous garantira qu'un sous-programme est exécuté chaque fois qu'on appuie deux fois sur ESC. Dans le cas présent, vous pouvez faire en sorte que le sous-programme affiche les valeurs des variables qui vous intéressent, vous permettant d'en modifier par INPUT, et puis retourne à l'exécution !

ESC utilisé une seule fois peut être très utile pour mettre au point un programme graphique. Quand vous appuyez sur ESC, les effets graphiques (sauf les caractères clignotants) s'arrêtent au point où ils en sont, si bien que vous pouvez utiliser le procédé pour voir dans quel ordre les choses se passent. Appuyer sur une autre touche fait reprendre l'action, et il n'y a pas de limite au nombre de fois que vous pouvez appuyer sur ESC pour vérifier l'évolution de l'image. Si cela ne suffit pas, ajoutez une boucle de pause temporairement dans votre programme graphique, et exécutez-le au ralenti, en utilisant ESC pour vérifier les parties délicates.

Suivre les boucles

Un programme peut se montrer déconcertant en s'exécutant sans message d'erreur, mais sans le résultat attendu non plus ! C'est en fait le symptôme d'une erreur de préparation, mais quelquefois il s'agit d'une simple inadvertance, et la procédure ON BREAK GOSUB peut rendre de grands services pour débusquer une erreur, puisqu'elle vous permet d'afficher l'état des variables à n'importe quel stade du pro-

gramme, puis de continuer l'exécution. Quelquefois cependant, vous souhaitez avoir une forme plus simple de dépistage. Si votre programme contient un grand nombre de lignes IF...THEN...ELSE, il arrive souvent que l'une d'elles ne fasse pas ce que vous en attendez. En pareil cas, le CPC464 vous donne de l'aide avec deux instructions TRON et TROFF.

TRON (comment vous figuriez-vous qu'on avait trouvé le nom du film ?) signifie TRACE ON (Dépistage en action), et son effet est d'imprimer sur l'écran les numéros de ligne au moment de leur exécution. Les numéros de ligne sont affichés entre crochets, au début d'une ligne d'écran, et avant tout affichage dû au programme. Essayez de taper TRON et exécutez ensuite un programme qui tourne lentement. TRON est particulièrement efficace si vous n'êtes pas sûr de ce que fait le programme, et cela peut être très commode pour désigner ce qui ne va pas dans une boucle. Souvenez-vous que vous pouvez combiner TRON avec d'autres instructions de dépistage d'erreur. Vous pouvez par exemple arrêter le programme, modifier des variables, et puis continuer, avec TRON pour montrer quelles lignes sont exécutées. Taper TROFF (puis ENTER) arrête le processus d'affichage des numéros de ligne.

Appendice B

Sortie sur imprimante

Le CPC464 est délicieusement facile à connecter à n'importe quelle imprimante utilisant la *sortie parallèle Centronics*. Il vous faudra un câble de connection approprié, qui vous sera fourni par le magasin qui vous a vendu le CPC464. L'une des extrémités du câble se branche sur la prise PRINTER de l'ordinateur, et l'autre se glisse aisément dans la prise de l'imprimante.

Le grand avantage de ce système est qu'il vous laisse le choix de l'imprimante que vous voulez utiliser. Vous pouvez par exemple choisir l'Epson RX-80, très répandue et facilement adaptable à de nombreux ordinateurs, ou une imprimante à marguerite pour une meilleure qualité d'impression, ou, pour le tracé de courbes, une table traçante comme l'étonnante petite imprimante/traçante Tandy CGP-115.

Vous n'êtes pas limité, comme avec d'autres marques d'ordinateurs, à utiliser l'imprimante « maison », et grâce à la sortie Centronics, vous avez le choix entre des imprimantes de tous les prix.

Selon le manuel, le CPC464 envoie deux codes à la fin de chaque ligne. L'un est le code de saut de ligne, ASCII 10, et l'autre le code de retour chariot, ASCII 13. La plupart des imprimantes courantes peuvent être « configurées » de manière à faire avancer le papier soit quand elles reçoivent le saut de ligne, soit quand elles reçoivent le retour chariot. On effectue en général la configuration en basculant un interrupteur à l'intérieur de l'imprimante.

Si vous constatez que votre ordinateur affiche tout sur la même ligne, ou envoie deux espaces entre les lignes, le remède usuel est de changer la position de cet interrupteur. Mais j'ai constaté que toutes mes imprimantes envoient deux espaces entre les lignes, quelle que soit la position des interrupteurs, ce qui paraît indiquer que le CPC464 envoie deux sauts de ligne. Si vous utilisez une Epson RX-80, vous pouvez y remédier en mettant l'espacement de ligne à une valeur inférieure à la valeur normale de 7/72 pouces. L'opération est réalisée par l'instruction :

PRINT # 8,CHR$(27);« A » ;CHR$(7)

quand l'imprimante est allumée. Cela élimine le problème *à condition que le CPC464 et l'imprimante utilisent le même nombre de caractères par ligne.* Si vous fixez le nombre de caractères par ligne à 40 pour l'imprimante, comme cela a été fait dans ce livre, vous devrez en faire autant sur l'ordinateur en utilisant WIDTH 40. Si ce n'est pas fait, quand une ligne déborde sur la ligne de papier suivante, l'espacement sera trop faible. Heureusement, la plupart des imprimantes permettent cet ajustement d'espacement de ligne, mais vous devrez consulter le manuel de votre imprimante si vous utilisez une autre marque.

Appendice C

Les trucs des touches

Un trait essentiel des ordinateurs modernes est l'existence de « touches programmables ». Cela signifie qu'une série de touches peut être programmée à donner des effets choisis par l'utilisateur. Au lieu de taper LIST (ENTER) à chaque fois que vous voulez un listing, par exemple, vous pouvez consacrer une touche à ce rôle, ou bien vous pouvez avoir une touche qui vous donne PRINT TAB(2) « chaque fois que vous en avez besoin pour écrire du texte.

Bien qu'un certain nombre d'ordinateurs consacrent des touches à ces fonctions, très peu offrent une manière simple de les programmer. Certains utilisateurs en viennent même à soupçonner que leurs « touches programmables » sont à peine plus qu'un ornement ! Le CPC464 vous permet de définir des actions pour jusqu'à 32 touches, utilisant des instructions BASIC simples.

Il y a toutefois des restrictions pratiques. Les codes d'instruction produits par l'action de chaque touche ne doivent pas excéder 32 caractères par touche. Le total de ces caractères ne doit pas excéder 120. En fait, il est peu probable que vous trouverez ces limitations gênantes.

Supposez donc que vous vouliez redéfinir une touche. Cela n'a de sens que si c'est une touche dont vous n'allez pas vous servir au cours de votre travail. Vous pouvez utiliser les codes ASCII 128 à 159 à cet effet, et les treize premiers codes sont déjà alloués à des touches, celles du bloc numérique du côté droit du clavier. L'appendice III, page 15 du manuel du CPC464 montre les codes de ces touches, et vous verrez que la

touche 0 est allouée au code 128. Pour lui faire donner LIST (ENTER) vous devez programmer comme suit :

KEY 128,"LIST"+CHR$(13)

et appuyer sur ENTER. L'ENTER de la fin de l'instruction est produit par le CHR$(13), dans la définition. Quand vous appuyez sur le 0 du bloc numérique, vous voyez maintenant le listing de votre programme.

Supposez que vous preniez un code ASCII comme 156, qui n'a pas de touche au clavier. Essayez, tapez :

KEY 156,"PRINT:PRINT"+CHR$(13)

Ensuite vous pouvez faire qu'*une autre touche* donne le code ASCII 156. Prenez le crochet carré qui se trouve au-dessus de la touche ENTER. Tapez.

KEY DEF 17,1,156

et appuyez sur ENTER. Maintenant quand vous appuyez sur cette touche, votre PRINT:PRINT apparaît et s'exécute. Le 17 dans l'instruction est le numéro d'INKEY pour cette touche, si bien que n'importe quelle touche peut être redéfinie ainsi. Toutes ces définitions disparaîtront quand la machine sera réinitialisée par SHIFT CTRL ESC. Si vous ne voulez pas que l'instruction soit exécutée dès qu'on presse la touche, omettez la partie CHR$(13). Quand vous utilisez PRINT TAB(, par exemple, vous voudrez ajouter le numéro de colonne, et la parenthèse fermante, puis du texte.

Appendice D

Piégeage d'erreurs

Plus tôt dans ce livre, nous avons rencontré l'idée de *piège à inepties*. Il s'agit d'une méthode de vérification des données introduites au clavier, pour voir si elles sont conformes à ce qui est acceptable ou non. On effectue le tri en utilisant des lignes comme :

60 IF LEN(A$)=0 THEN GOTO 1000:GOTO 50

et vous devez avoir un type de piège séparé pour chaque erreur possible. Cela peut être très fastidieux, et il arrive souvent que vous ne prévoyiez pas une autre erreur. Le CPC464 est une des rares machines qui vous offre une instruction différente de piégeage d'erreur ; ON ERROR GOTO.

```
10 ON ERROR GOTO 1000
20 PRINT "Tapez un mot"
30 INPUT A$
40 L=LEN(A$)
50 PRINT 1/L
60 END
1000 PRINT"Le mot n'a aucune lettre !"
1010 RESUME 20
```

Fig. D.1. Utilisation de l'instruction ON ERROR GOTO.

La *figure D.1* vous donne un exemple très artificiel, un exemple réaliste prendrait trop de temps à taper. Dans cet exemple, la longueur d'un mot est mesurée, et le nombre est inversé (utilisé pour diviser 1). C'est impossible si la longueur est 0, et le ON ERROR GOTO est conçu pour piéger cela. Vous pourriez avoir une entrée de zéro, par exemple, en tapant ENTER avant d'avoir appuyé une autre touche. Maintenant,

normalement, si cela arrivait, vous auriez un message d'erreur et le programme s'arrêterait. Le grand avantage d'ON ERROR GOTO est que le programme ne s'arrête pas, quand une erreur est trouvée ; à la place le programme se déroute vers le sous-programme. Dans cet exemple, le sous-programme affiche un message, puis regagne la ligne 20. C'est délicieusement simple, mais il faut faire quelques expériences. Vous voyez, si votre programme contient encore des erreurs de syntaxe, elles déclencheront aussi le sous-programme, et cela peut amener à une conduite déconcertante si le programme saute brusquement à une ligne différente.

```
10 CLS
20 ON ERROR GOTO 1000
30 FOR N=1 TO 5
40 READ X:Y$=STR$(SQR(X))
50 PRINT "Nombre :";X;"  Racine carrée :
";Y$
60 NEXT
70 DATA 5,4,3,-2,2
80 END
1000 Y$=STR$(SQR(ABS(X)))+"I"

1010 RESUME 50
```

Fig. D.2. Un autre exemple de vérification d'erreur, avec retour automatique.

La *figure D.2* montre un autre exemple. La ligne 20 s'assure que toute erreur enverra le programme à la ligne 1000. Le programme lit alors un ensemble de nombres et forme une chaîne à partir de la racine carrée de chaque nombre. Le problème est qu'il y a un nombre négatif. Or, l'ordinateur ne peut trouver la racine carrée d'un nombre négatif, parce qu'il s'agit d'un nombre « imaginaire ». Mettre au carré un nombre négatif ou positif donne toujours un nombre positif, aussi n'y a-t-il pas de nombre réel dont le carré soit négatif. Toutefois, dans beaucoup d'applications on assigne un sens à cette expression, par exemple pour exprimer une longueur mesurée à angle droit par rapport à une autre.

Quand on lit —2 l'effet de la ligne 20 est d'envoyer le programme à la ligne 1000, puisque l'ordinateur diagnostique une erreur. A la ligne 1000 on prend la valeur absolue de —2, et on extrait la racine carrée. On ajoute la lettre I pour indi-

quer qu'il s'agissait d'un nombre négatif et que la racine est imaginaire (ou « complexe »). La ligne 1010 renvoie au déroulement normal, de sorte que le message, avec la nouvelle valeur de Y$ est affiché.

```
10 CLS
20 ON ERROR GOTO 1000
30 FOR N=1 TO 5
40 READ X:Y$=STR$(SQR(X))
50 PRINT "Nombre :";X;"  Racine carrée :
";Y$
60 NEXT
70 DATA 5,4,3,-2,2
80 END
1000 Y$=STR$(SQR(ABS(X)))+"I"
1005 PRINT "Ligne ";ERL;" --Erreur";ERR
1010 RESUME 50
```

Fig. D.3. Affichage du numéro de ligne et de l'erreur dans un sous-programme de traitement d'erreur.

La *figure D.3* montre une variante sur ce thème, qui vous fait savoir quelle erreur s'est produite, et à quelle ligne. A la ligne 1005, ERL donne le numéro de ligne de l'erreur, et ERR le numéro de code de l'erreur. Vous trouverez la liste des codes d'erreur dans le manuel du CPC464. C'est une ligne commode à utiliser quand vous testez un programme, parce que si l'erreur n'est pas une de celles prévues, vous pourrez le constater en lisant l'écran.

Utilisation de PRINT

On peut désormais examiner une instruction qu'il était trop compliqué de détailler auparavant. Il s'agit de PRINT USING. Le but de cette instruction est d'obliger l'affichage à respecter une certaine mise en forme. Après USING il doit y avoir une chaîne, qui peut être déclarée auparavant. Cette chaîne doit prendre une forme appropriée à ce que vous voulez afficher.

Tout cela a l'air bien mystérieux, aussi reportez-vous à la *figure D.4*. Dans cet exemple, A$ est défini comme

#. # #. Cela signifie que n'importe quel nombre qu'on affiche par PRINT USING A$ sera affiché avec trois chiffres avant le point décimal et deux après. En informatique, A$ est appelé une chaîne de « format ».

```
10 CLS
20 PRINT TAB(15)"PRINT USING"
30 PRINT:PRINT
40 N=140.2716
50 PRINT "N VAUT";N
60 A$="###.##"
70 PRINT "Avec PRINT USING, N vaut ";USI
NG A$;N
80 PRINT "La T.V.A sur #";N;" est de #";
15*N/100
90 PRINT "C'est plus propre sous la form
e #";USING A$;N*15/100
```

Fig. D.4. Un exemple simple de PRINT USING.

C'est de loin l'usage le plus utile de PRINT USING, mais le manuel en énumère d'autres. Certains n'offrent d'intérêt que si vous affichez des valeurs en dollars (et pourtant ce BASIC a été écrit en Grande-Bretagne) !

Index

Table des matières

L'impression de ce livre
a été réalisée sur les presses
des Imprimeries Aubin
à Poitiers/Ligugé



Achevé d'imprimer en mars 1985
Dépôt légal 0214-03-1985 — N° d'impression, L 19735
Collection 50 — Édition 01

Imprimé en France

MICROPRATIQUES

L'utilisation de l'Amstrad CPC464

Hachette *Informatique*

L'Amstrad CPC 464 est un ordinateur aux performances exceptionnelles. Mais pour pouvoir exploiter toutes ses possibilités, il manquait un ouvrage de complément à l'excellent manuel livré avec la machine.

"L'utilisation de l'Amstrad" vous apprend à vous familiariser avec le fonctionnement du CPC 464, vous initie à la programmation dans sa version avancée du BASIC et vous donne les moyens de concevoir vos propres programmes adaptés à vos besoins.

Voici un livre qui a été élaboré en ayant la machine en main. Toutes ses possibilités originales y sont largement développées et commentées : depuis les instructions BASIC spécifiques jusqu'à la gestion des fenêtres, des interruptions, du processeur sonore.



F 125.00/85/3

10/0044/7

Imprimé en France
SUD-OFFSET - 94 RUNGIS

Design Contours.
Illustration : Agnès Lévy.


L'utilisation de l'Amstrad CPC464
Hachette



Document numérisé avec amour par :

AMSTRAD CPC

MÉMOIRE ÉCRITE

https://acpc.me/